Reasoning About DNSSEC

A thesis submitted in partial fulfillment of the
requirements for the award of degree of

Master Of Technology

in
Artificial Intelligence

by

S Girikumar

Department Of Computer And Information Sciences

UNIVERSITY OF HYDERABAD

Hyderabad, India

June 2009

CERTIFICATE

This is to certify that the project work entitled "Reasoning About DNSSEC" being submitted to University of Hyderabad by Sajja Girikumar (Reg. No. 07MCMI12), in partial fulfillment for the award of the degree of Master of Technology in Artificial Intelligence, is a bonade work carried out by him under my supervision.

Dr Vineet C.P Nair	Mr. Wilson Naik Rathore B
Project Supervisor	Project Supervisor
Department Of CIS	Department Of CIS
University Of Hyderabad	University Of Hyderabad

Head Of The Dept.	Dean
Department Of CIS	School Of MCIS
University Of Hyderabad	University Of Hyderabad

"The secret of life is not to do what you like, but to like what you do."

Anonymous

Abstract

by S Girikumar

This thesis is concerned with developing a logic based framework so as to represent and reason about the DNSSEC (Domain Name System Security) protocol. DNSSEC provides security services to the existing DNS Protocol mainly through public key cryptography. But, it is well known that even the use of the most perfect cryptographic tools does not always ensure the desired security goals. This situation arises because of logical flaws in the design of protocols. Our aim is to formally specify and verify the DNSSEC protocol using modal logic so as to derive the correctness of the protocol.

Acknowledgements

I would like to express my sincere gratitude to Dr. Vineet Nair and Mr. Wilson Naik Rathore B, my project supervisors, for valuable suggestions and keen personal interest throughout the course of research. They encouraged, supported, corrected and guided me during the project. The project has been a learning and growing experience for me.

I express my heart-felt thanks to Rajender Kumar for his help and advises during my M.Tech. I thank S Chetan and Neelakantam for their moral support, I am also extremely thankful to all lab mates for their countless advises and discussions with me. I would also like to thank the Open Source Community who provided the free software and documentation to my work.

I am extremely grateful to our Head of the Department, Prof. Arun Agarwal, for providing excellent computing facilities and a conclusive atmosphere for accomplishing my project. I convey my heartfelt thanks to AI Lab staff all with out whose help this project would not have seen light.

I would like to take this opportunity to thank my friends who have been morale boosters for me, encouraged me to take up this course and supported me throughout this course with their love and affection. My special thanks to my wonderful parents who have always supported me in all my decisions.

Sajja Girikumar

Contents

List of Figures

Abbreviations

BAN	**B**urrows **A**badi**N**eedhem
AT	**A**badi **T**uttle
GNY	**G**ong **N**eedham **Y**ahalom
SVO	**S**yverson **V**an **O**orschot
DNS	**D**omain **N**ame **S**ystem
DNSSEC	**D**omain **N**ame **S**ystem **SEC**urity

To My Parents

Chapter 1

Introduction

Security protocols are widely used for providing security services in different distributed systems. Deficiencies in these protocol design will result in negative consequences over the system they are supposed to protect. The protocols need to be specified and verified before it can be put into use. Burrows, Abadi, and Needham have proposed a logic called BAN logic [1] specially designed for the analysis of authentication protocols. The logic provides a language for describing the beliefs of the parties involved in a protocol, and a set of inference rules that describe how these beliefs evolve as a result of communication.

The successfull usage of BAN logic in verifying security protocols opened a new way of verifying security protocols using Modal logic. Many other Modal logic Based systems like AT, GNY, VO and SVO were also proposed and used for specification and verification of security protocols. The formal Analysis of a protocol in Modal Logics begins by formalizing the message exchanges of the protocol in the language of its logic first. Then all the initial assumptions of the protocol and assertions about each message exchange are written, Finally, Axioms of the particular modal system are applied on the assumptions and the assertions in order to derive the conclusion.

In this project we develop a framework for are formally specifying and verifying the DNSSEC with SVO Logic which is most suitable for representing and reasoning about DNSSEC. DNSSEC is a security extensions to the existing Domain Name System which is used for translation between name and IP addresses. The DNSSEC is defined to achieve Integrity and Origin Authentication security goals. That is through DNSSEC the user can have trust over the replies received DNS Server. The DNSSEC uses Public Key Infrastructure to provide security services to the DNSSEC.

1.1 Motivation

Designing security protocols is a difficult task, because small flaws will open big gates for attackers to breach the system. So the security protocols need to undergo verification to uncover the flaws before they are going to use. Formal logic based approach to specify and verify is a simple and widely used technique in security protocol verification techniques. Many logic based techniques were designed to verify protocols.

Domain Name Systems is central to the internet. The flaws in this system will cheat endusers, for example the attacker can redirect an enduser to fake sites instead of actual one by masquerading DNS server. To protect DNS system DNSSEC is introduced. DNSSEC is designed for security purposes it need to be verified carefully. Through our project we are verifying DNSSEC using a formal approach.

1.2 Organization

Organization of chapters are as follows: Chapter 2 discuses about related work which discuses BAN [1], GNY [2], AT [3], VO [4], SVO [5], and few security protocols that have used SVO as the formal framework. Chapter 3 discuses what is DNSSEC and some basics about DNS. chapter 4 discuses about the modal logic system SVO. In chapter 5 we extend SVO and show how it can be successfully used in analyzing DNSSEC. In chapter 6 we provide implementation details. We conclude with chapter 7.

Chapter 2

Related Work

Formal analysis is a widely accepted method to check the correctness of a security protocol and uncover its potential flaws. Researchers have developed numerous formal techniques and tools and successfully applied them to either verify a security property or to catch a flaw in a security protocol. BAN (Burrows, Abadi and Needham) logic is the first highly successful formal specification and verification tool. In this chapter we will discuss about BAN logic and its successors (AT, GNY, VO, SVO). There after we will discuss about how SVO is successfully used in protocols like GSI [6] and Hidden ID [7]

2.1 BAN Logic

BAN [1] is a logic of belief which was the first attempt in applying modal logic to formally specify and verify security protocols. Since security protocols involve principals sending and receiving messages to each other, each principal holds certain beliefs about these messages. For example, a principal A believes that a message M is fresh. In BAN notations, it is represented by A believes Fresh(M). The following sections discuses basic notions, rules and formalizing of protocols of BAN logic. These Constructs are clearly represented in [1].

2.1.1 BAN Notions

—**P *believes* X :** P may act as if X is true.

—**P *sees* X :**Someone has sent a message containing X to P, who can read and repeat X (possibly after doing some decryption).

—***said* X :** P once said X. The principal P at some time sent a message including the statement X.

—**P *controls* X :** P has jurisdiction over X. The principal P is an authority on X and should be trusted on this matter.

—***fresh*(X) :** The formula X is fresh; that is, X has not been sent in a message at any time before the current run of the protocol.

—**P** $\xleftrightarrow{K}$ **Q :** P and Q uses the shared key K to communicate.

— $\overset{K}{\longmapsto}$ **P :** P has K as a public key. The matching secret key (denoted K^{-1}) will never be discovered by any principal except P or a principal trusted by P.

—**P** $\overset{K}{\rightleftharpoons}$ **Q:**The formula X is a secret known only to P and Q

—$\{X\}_K$**:** This represents the formula X encrypted under the key K.

—$\langle X\rangle_K$**:** This represents X combined with the formula Y.

—**Nonce:** In security engineering, a nonce stands for number used once. It is often a random or pseudo-random number issued in an authentication protocol to ensure that old communications cannot be reused in replay attacks.

2.1.2 BAN Rules

This section discuses the major rules of the BAN logic used in formal verification of protocols.

1. **Message Meaning Rules**

   1.1. $\dfrac{\text{P } \textit{believes} \text{ Q} \xleftrightarrow{K} \text{P, P } \textit{sees}\{X\}_K}{\text{P } \textit{believes} \text{ Q } \textit{said} \text{ X}}$

   That is, if P believes that the key K is shared with Q and sees X encrypted under K, then P believes that Q once said X. For this rule to be sound, we must guarantee that P did not send the message himself; it suffices to recall that $\{X\}_K$ stands for a formula of the form $\{X\}_K$from R, and to require that $R \neq P$. Similarly,

   1.2. $\dfrac{\text{P } \textit{believes} \overset{K}{\longmapsto} \text{Q, P } \textit{sees}\{X\}_{K-1}}{\text{P } \textit{believes} \text{ Q } \textit{said} \text{ X}}$

That is, if P believes that the key K is public key of Q and sees X encrypted under private key K^{-1}, then P believes that Q once said X.

1.3. $\dfrac{\text{P } \textit{believes} \text{ Q} \overset{K}{\rightleftharpoons} \text{P, P } \textit{sees} \langle X \rangle_K}{\text{P } \textit{believes} \text{ Q } \textit{said} \text{ X}}$

If P believes that the secret Y is shared with Q and sees $\langle X \rangle_Y$, then P believes that Q once said X.

2. **Nonce Verification Rule** In BAN, the freshness of nonce can be verified as,

2.1. $\dfrac{\text{P } \textit{believes fresh}\text{(X),P } \textit{believes} \text{ Q } \textit{said} \text{ X}}{\text{P } \textit{believes} \text{ Q } \textit{believes} \text{ X}}$

That is, if P believes that X could have been uttered only recently and that Q once said X , then P believes that Q believes X.

3. **Jurisdiction Rule**

3.1. $\dfrac{\text{P } \textit{belives} \text{ Q } \textit{controls} \text{ X, P } \textit{believes} \text{ Q } \textit{believes} \text{ X}}{\text{P } \textit{believes} \text{ X}}$

If P believes Q have control over X and P believes that Q believes X then P believes X.

4. If a Principal sees or believers a formula then it can see or believe its components also.

4.1. $\dfrac{\text{P } \textit{sees} \text{ (X,Y)}}{\text{P } \textit{sees} \text{ X}}$, 4.2. $\dfrac{\text{P } \textit{sees} \langle X \rangle_Y}{\text{P } \textit{sees} \text{ X}}$, 4.3. $\dfrac{\text{P } \textit{believes} \text{ Q} \overset{K}{\longleftrightarrow} \text{P, P } \textit{sees} \{X\}_K}{\text{P } \textit{sees} \text{ X}}$

4.4. $\dfrac{\text{P } \textit{believes} \overset{K}{\longmapsto} \text{P, P } \textit{sees} \{X\}_K}{\text{P } \textit{sees} \text{ X}}$, 4.5. $\dfrac{\text{P } \textit{believes} \overset{K}{\longmapsto} \text{Q, P } \textit{sees} \{X\}_{K-1}}{\text{P } \textit{sees} \text{ X}}$

5. If one part of formula is fresh then entire formula is also fresh

5.1. $\dfrac{\text{P } \textit{believes fresh}\text{(X)}}{\text{P } \textit{believes fresh}\text{(X,Y)}}$.

2.1.3 Formalizing Security Protocols Using BAN Logic

This section explains how protocols can be formalized and verified using BAN logic using the example, taken form [8]. BAN uses following four steps in analysing security protocols.

1. Idealise the protocol.

2. State all the initial assumptions of the protocol.

3. For each message transmission in the protocol of the form A $\longrightarrow$ B : M, write assertion of the form: B *received* M.

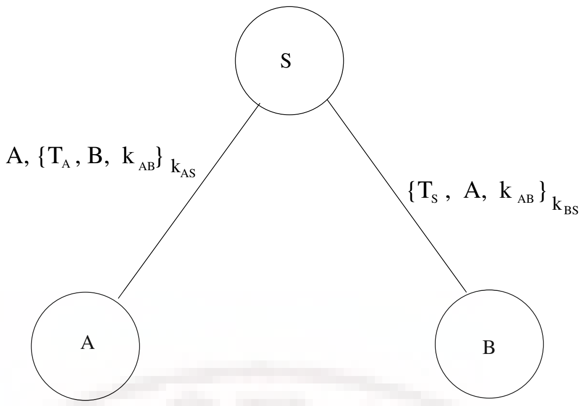


FIGURE 2.1: Wide Mouthed Protocol

4. Apply BAN rules on the assumptions and assertions to derive beliefs held by other principals of the protocol.

The example given bellow explains how BAN rules are applied in verifying the Wide Mouth Frog Protocol. The Wide Mouth Frog protocol is a computer network authentication protocol designed for use on insecure networks. It allows individuals communicating over a network to prove their identity to each other while also preventing eavesdropping or replay attacks, and provides for detection of modification and the prevention of unauthorized reading. This can be proven using BAN logic. The protocol comprises of two steps as shown in Figure 2.1. In the first message, a principal A sends a session key k_{AB} along with a timestamp T_a to S. After checking the timeliness of the first message, S adds its own timestamp and sends the second message to B.

Step1: Wide mouthed Protocol can be Idealized as given below

Message1: $A \longrightarrow S : \{\, T_A,\ (A \overset{K_{AB}}{\longleftrightarrow} B)\}_{K_{AS}}$

Message2: $S \longrightarrow B\{T_S,\ A\ believes\ (A \overset{K_{AB}}{\longleftrightarrow} B)\}_{K_{BS}}$

Idealization of protocol is not only understanding the protocol but also need to understand the believes corresponding to the messages received and sent. i.e in Message2 the server adds its belief on shared key$A \overset{AB}{\longleftrightarrow} B$

Step2: Write the initial assumptions in the protocol. The wide mouth frog protocol have the following initial assumptions,

$A\ believes\ A \overset{K_{AS}}{\longleftrightarrow} S$ $\qquad B\ believes\ B \overset{K_{BS}}{\longleftrightarrow} S$

$S\ believes\ A \overset{K_{AS}}{\longleftrightarrow} S \qquad S\ believes\ B \overset{K_{BS}}{\longleftrightarrow} S$

$S\ believes\ fresh(T_A) \qquad B\ believes\ fresh(T_S)$

$B\ believes\ A\ controls\ A \overset{K_{AB}}{\longleftrightarrow} B$

$B\ believes\ (S\ controls\ (A\ believes\ A \overset{K_{AB}}{\longleftrightarrow} B))$ i.e Initially principals A and S believes $A \overset{AS}{\longleftrightarrow} S$ as good communication key for A and S, principals B and S believes $A \overset{BS}{\longleftrightarrow} S$ as good communication key for B and S, B believes principal A controls the communication key between A and B, and finally B believes that S have the control over A's belief on sahred key $A \overset{AB}{\longleftrightarrow} B$

Step3: Write assertions on messages received and sent, In wide mouth frog protocol assertions can be made as

$S\ received\ \{T_A,\ (A \overset{K_{AB}}{\longleftrightarrow} B)\}_{K_{AS}}$

$B\ received\ \{T_S,\ A\ believes(A \overset{K_{AB}}{\longleftrightarrow} B)\}_{K_{BS}}$

The above assretions represents that S received communication for principals A and B, and time stamp from principal A. And B received time stamp and communication key that is believed by principal A.

Step4: Derive new rules based on BAN rules, Assertions and from initial assumptions, In our example wide mouth frog protocol the BAN logic will following conclusions

$S\ believes\ A\ believes\ A \overset{K_{AB}}{\longleftrightarrow} B$

$A\ believes\ A \overset{K_{AB}}{\longleftrightarrow} B$

$B\ believes\ A \overset{K_{AB}}{\longleftrightarrow} B$

$B\ believes\ A\ believes\ A \overset{K_{AB}}{\longleftrightarrow} B$

From the derived rules we can conclude that the BAN has derived security goals of wide mouth frog protocol.

2.2 BAN Extensions

With the success of BAN logic, reaserchers realized the importance of logic in formal specification and verification of authentication protocols. Eventhough BAN has identified flaws in many security protocols, it has been seriously analyzed for finding potential weaknesses and corrections were suggested. In this way many new modal logic based systems were proposed. GNY [2, 9] was the immediate successor of BAN which extended BAN with few additions which offers important advantages over BAN. GNY uses basic notions of BAN, and expanded this set with few more notions like `recognizability` which captures a recipients expectation of the contents of messages he receives which can help in incorporating additional properties of messages in the reasoning process. The notion of `possession` incorporated in this assumes that principals can include in messages data they do not believe in, but possess, and also The `not-originated-here` notion allows us to determine that certain messages are not replays of a recipients own previous messages in a session. This makes GNY to analyze much wider range of protocols. The Next successor AT, Abadi and Tuttle [3] extended BAN by providing new semantics to the logic of BAN. They have reformulated the BAN logic, called AT logic, and proved their axiomatization to be sound with respect to their semantics.

Paul C. van Oorschot [4] designed a new logic which extended GNY logic and particularly created for reasoning about protocols which is based on Deffie-Hellman type key agreement. And this was extended as SVO logic by Paul C. van Oorschot and Syverson. The detailed discussion about SVO is presented in chapter4. The next section of this chapter discuses how SVO is used in analysing some security protocols. Before going in to the next chapter we would like to point out two example systems in literature wherein SVO logic has been used as the underlying formal tool. Hidden-ID Authentication is a Authentication scheme in which the server will identify the user by exchanging messages encrypted by the secret keys which is generated by the client earlier and stored at server. In [7], the system was verified by SVO logic for its correctness successfully. The GSI (Grid Security Infrastructure) Protocol is a collection of security services designed to secure the grid. In [6] the SVO logic is successfully used in verifying its correctness.

Chapter 3

DNSSEC

3.1 Overview Of DNS

The Domain Name System (DNS) is a hierarchically distributed database that provides information fundamental to Internet operations, such as translating between human readable host names and Internet Protocol (IP) addresses known as name resolution or simply resolution. A client entity known as a Resolver submits queries to, and receives responses from, the DNS server. The responses contain Resource Records (RRs) containing the desired name-address resolution information. There are two modes of resolution in DNS: iterative and recursive. In the iterative mode, when a name server receives a query for which it does not know the answer, the server will refer the querier to other servers that are more likely to know the answer. Figure 3.1 shows how dns resolution happens in iterative mode. In recursive mode if server does not know the answer, the server will forward that dns request to its near DNS servers, and fetches name resolution from its neighbor servers. Figure 3.2 shows how dns resolution happens in iterative mode. The details about DNS and DNSSEC are clearly presented in [10–12]

3.2 DNSSEC

Users accessing hosts on the Internet rely on the correct translation of host names to IP addresses which can be served by DNS servers. Because of the importance of the information served by DNS server, there is a strong demand for authenticating the server and the information served by that server in DNS system. The current DNS does not prevent attackers from modifying or injecting DNS messages. A typical

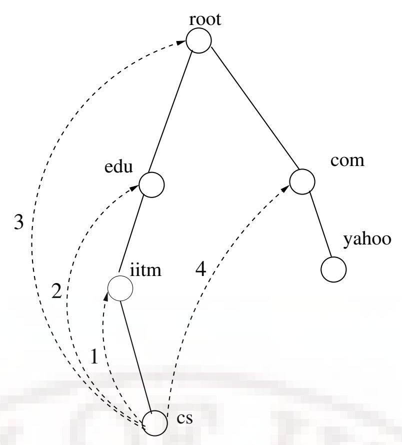


FIGURE 3.1: Iterative Naming Resolution

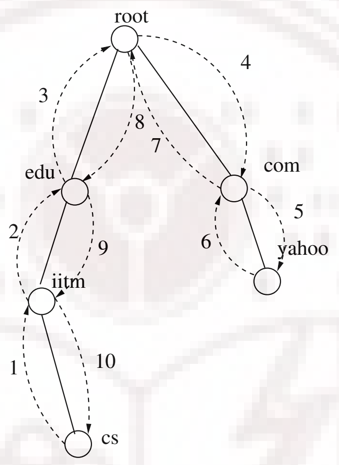


FIGURE 3.2: Recursive Naming Resolution

attack, referred to as DNS spoofing, allows an attacker to manipulate DNS answers on their way to the users. If an attacker makes changes in the DNS tables of a single server, those changes will propagate across the Internet. Some of typical attacks on DNS are explained in [13] and [14] To stop these attacks on DNS system it is necessary to add security to the existing DNS System.To do this Domain Name System Security Extensions were defined. Securing DNS means providing data origin authentication (data is received from a party from which it was sent) and integrity (data is same as it was sent and not modified in between) to the DNS system. Confidentiality is not required, since the information stored in the DNS database is supposedly public anyone in the internet can see the DNS replies. All answers from DNSSEC servers are digitally signed. By checking the signature, a DNS client is able to check if the information originated from a legitimate server and that data is identical to the data on the DNS server.

To achieve Security in Domain Name System, DNSSEC adds four records namely Resource Record Signature (RRSIG), DNS Public Key (DNSKEY), Delegation Signer (DS), and Next Secure (NSEC) to the existing DNS System.
DNSKEY : DNSSEC uses public key cryptography to sign and authenticate DNS resource record sets (RRsets). The public keys are stored in DNSKEY resource records and are used in the DNSSEC authentication process
RRSIG : The digital signatures of DNS replies are stored in RRSIG record. RRSIG value is created by encrypting the hash value of the DNS reply with private key of the zone.
DS : Contains the hash value of a child zone's DNSKEY, which is needed in authenticating client zone DNSKEY.
NSEC : If a DNS Server in a zone doesn't have any authoritative data to return then NSEC will contain deligation point to the next authoritative server which contains some authoritative data return or deligation point to some other server.

The DNS server in a zone creates public key, private key pair and publishes the public key in zone and keeps private key as secrete. Private key is used to create signature, and the public key is used to verify the signature. There will be a single private key that signs a zones data, but multiple keys are possible. For example, there may be keys for each of several different digital signature algorithms. If a security aware resolver learns a zones public key, it can authenticate that zones signed data. Security-aware resolvers authenticate zone information by forming an authentication chain from a newly learned public key back to a previously known authentication public key,which in turn must have been learned and verified previously. An alternating sequence of DNS public key (DNSKEY) RRsets and Delegation Signer (DS) RRsets forms a chain of trust. A DNSKEY RR is used to verify the signature covering a DS RR and allows the DS RR to be authenticated. The DS RR contains a hash of another DNSKEY RR and this new DNSKEY RR is authenticated by matching the hash in the DS RR. This new DNSKEY RR in turn authenticates another DNSKEY RRset and, in turn, some DNSKEY RR in this set may be used to authenticate another DS RR, and so forth until the chain finally ends with a DNSKEY RR whose corresponding private key signs the desired DNS data. For example, the root DNSKEY RRset can be used to authenticate the DS RRset for "google.". The "google." DS RRset contains a hash that matches some "google." DNSKEY, and this DNSKEY's corresponding private key signs the "google." RRset. Private key counterparts of the "google." DNSKEY RRset sign data records such as "www.google." and DS RRs for delegations such as "subzone.google."

3.2.1 How DNSSEC Works

When Client send request to DNS Server, DNSSEC adds additional data to the DNS protocol responses that provide additional information (RRSIG,DNSKEY) to allow the DNS client to authenticate the RRset data response. The client can take the RRset response and use the algorithm referenced in the RRSIG record to generate the hash of the data. The RRSIG value can be encrypted using the DNSKEY public key which will, in effect, decrypt the hash in the RRSIG record. This operation allows the client to check that the hash of the RRset data matches the decrypted RRSIG hash. The DNSKEY would normally be provided as part of the additional section of a DNSSEC response. If the client has not validated the DNSKEY within some locally defined period, then the client should also validate the DNSKEY value. This entails verifying the RRSIG record on the DNSKEY value, using the same procedure as for the RRset validation.

However domain zone key validation also entails the construction of a trust chain back to a trust anchor point. If this domain key is not already a trust anchor then the client needs to query the parent zone for the DS record of the child zone, which returns both a public key value and an RRSIG value, and a DNSKEY RR. This public key needs to be validated using the DNSKEY of the parent zone. This parent zone public key, in turn, must be validated. This iterative process constructs a trust chain that, hopefully, leads back to a trust anchor. At that point the DNS response can be considered to be valided.

Figure 3.3 explains how the authentication chain will be established using DS RRset and DNSKEY RRset. Consider that the host "client.uohyd.ernet.in." in uohyd domain has made a DNS Request for "yahoo.com." whose RRset is available in "com" domain. The resolution process has followed these steps

- M$_{1D}$: client $\longrightarrow$ uohyd :$\langle DNSReq., yahoo.com.\rangle$
  `client` sends a request to its zone name server `uohyd` for DNS resolution.

- *M$_{1u}$: uohyd $\longrightarrow$ client* :$\langle DNSRep., yahoo.com., ernet\rangle$
  `uohyd` sends a message to client, to contact `ernet` for this particular naming resolution.

- *M$_{2D}$: client $\longrightarrow$ernet* :$\langle DNSReq., yahoo.com.\rangle$
  `client` sends a request to name server `ernet` for DNS resolution.

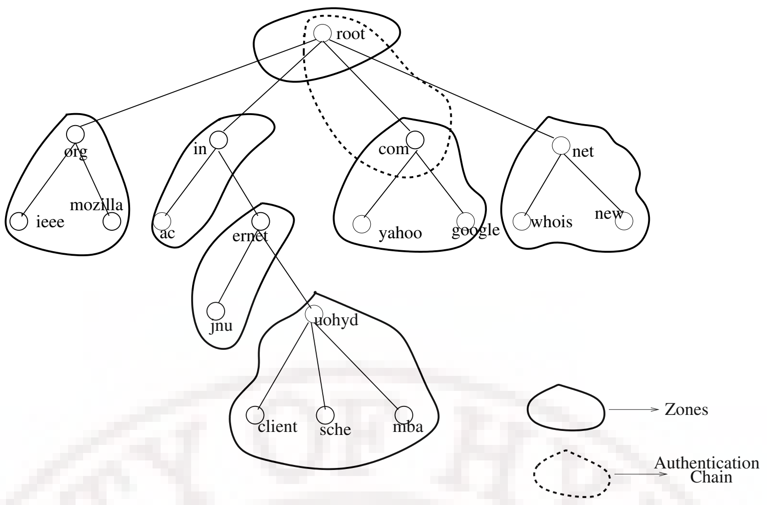


Figure 3.3: sample scenario

- M_{2r}*: ernet* $\longrightarrow$*client* :$\langle DNSRep., yahoo.com., in\rangle$

  `ernet` sends a message to client, to contact `in` for this particular naming resolution

- M_{3D}*: client* $\longrightarrow$*in* :$\langle DNSReq., yahoo.com.\rangle$

  `client` sends a request to name server `in` for DNS resolution.

- M_{3i}*: in* $\longrightarrow$*client* :$\langle DNSRep., yahoo.com., root\rangle$

  `in` sends a message to client, to contact `in` for this particular naming resolution.

- M_{4D}*: client* $\longrightarrow$*root* :$\langle DNSReq., yahoo.com.\rangle$

  `client` sends a request to name server `root` for DNS resolution.

- M_{4e}*: root* $\longrightarrow$*client* :$\langle DNSRep., yahoo.com., com\rangle$

  `root` sends a message to client, to contact `in` for this particular naming resolution

.

- M_{5D}*: client* $\longrightarrow$*com* :$\langle DNSReq., yahoo.com.\rangle$

  `client` sends a request to name server `com` for DNS resolution.

- M_{5u}*: com* $\longrightarrow$*client* :$\langle RRset_{yahoo.com.}, RRSIG_{D.com.}, DNSKEY_{com}\rangle$

  `client` receives DNS resolution response from `com`

The Authentication and Integrity of the DNS Resolution can be verified at client end by forming authentication chain as follows:

1. **Integrity Check:** Integrity will be achieved if the following condition is true

   - $hash(RRset_{uohyd}) == \{RRSIG_{uohyd}\} DNSKEY_{uohyd}$

   That is hash calculate at client is equal to the received hash value then the client will believe that received data is not modified in its way.

2. **Authentication:** Authentication will be achieved by establishing Authentication chain between the zone's with the help of DS RRset and DNSKEY RRset's.Consider in our example D.com. has already know the authentication publickey(DNSKEY) of root.To make newly learned DNSKEY as authentic it need to establish a authentication chain back to known DNSKY i.e root's DNSKEY. The steps given bellow shows how this chain will be formed

   - Verifying Sender DNSKEY(DNSKEY$_{com}$)

     $\{RRSIG(DS)\}_{DNSKEY_{root}} == DS_{com})$

   - Verifying Parent Zone DNSKEY(DNSKEY$_{root}$) The DNSKEY of root Zone is believed by every one it is not required to be verified and from this we can get authentication chain to the root node

Chapter 4

Modal Logic System - SVO

SVO is a logic for analyzing cryptographic protocols. This logic encompasses a unification of four of its predecessors in the BAN family of logics, namely GNY,AT, VO, and BAN itself. In this chapter we discuss about modal logic system SVO, and about its syntax and semantics.

4.1 The Language

This logic assume the existence of a set $\mathcal{T}_0$ of primitive terms, containing a number of disjoint sets of constant symbols representing principals, shared keys, public keys, private keys, numerical constants, etc. Building recursively on $\mathcal{T}$, have n-ary function symbols representing functions of n variables, for finite n , e.g. simple arithmetical functions, encryption, etc. In addition to these there are set of primitive proposition constants. These represent atomic propositions, which take the value true or false. The full set of terms is called $\mathcal{T}$. We actually require two formal languages, one for messages and one for formulae. Only formulae can be true or false or have principals beliefs attributed to them. On the other hand, some messages are not formulae, e.g., a message consisting of a name and a nonce. SVO is meant to encompass both languages.

In SVO the language of messages M_τ, is the smallest language over τ satisfying the following:

- X is a message if $X \in \tau$
- F(X1,.,Xn) a message if X1 , .. .,Xn are messages and F is any function.
- φ is a message if φ is a formula.

The language of formulae, M_τ, is the smallest language satisfies the following:

- p is a formula if p is a primitive proposition.
- $\neg\varphi$ and $\varphi \wedge \psi$, are formulae if φ and ψ are formulae
- P believes φ and P *controls* φ are formulae when φ is a formula and P is a principal,
- P *sees* X , P *received* X , P *says* X , P *said* X , and *fresh*(X) are formulae when X is a message and P is a principal,
- $P \overset{K}{\longleftrightarrow} Q$, $PK(P, K)$, and P has K are formulae when P and Q are principals and K is a key.

4.2 The Axioms of SVO logic

SVO logic has the following basic inference rules,

Modus Ponens: From φ *and*$\varphi \rightarrow \psi$ infer ψ

Necessitation: From $\vdash \varphi$ infer $\vdash$ P *believes* φ

Believing For any principal P and formulae ψ *and* φ

1. $P\ believes\ \varphi\ \wedge\ P\ believes\ (\varphi \rightarrow \psi) \rightarrow P\ believes\ \psi$
2. $P\ believes\ \varphi\ \rightarrow\ P\ believes\ (P\ believes\ \varphi)$

Axiom 1 says that a principal believes all that logically follows from his beliefs. Axioms 2 says that a principal can tell what he believes.

Source Association Keys are used to deduce the identity of the sender of a message,

3. $P \overset{K}{\longleftrightarrow} Q,\ \wedge\ R\ received\ \{X\}_K \rightarrow\ Q\ said\ X$
4. $PK_\sigma(Q, K)\ \wedge R\ received\ \{X\}_{K_{-1}} \rightarrow\ Q\ said\ X$

$PK_\sigma(Q, K)$ says that K is public signature verification key for Q.

Key Agreement this axiom is useful for Deffie-Hellman key exchange type of protocols

5. $((PK_\delta(P,K_p)) \wedge (PK_\delta(Q,K_q))) \;\rightarrow\; P \overset{K_{pq}}{\longleftrightarrow} Q$

Receiving A principal can receive a message from the concatenates of received messages and receives original message of encrypted message if it has the corresponding encryption key.

6. $P\ received\ (X_1, X_2, ..X_n) \;\rightarrow\; P\ received\ X_i$
7. $(P\ received\ \{X\}_K$ P has $\tilde{\text{K}}) \rightarrow\; P\ received\ X$

Where $\tilde{\text{K}}$ is decryption key corresponding to the encryption Key K

Seeing A principal sees anything he receives. A principal also sees all components of every message he sees and any message he can compute from what he sees. .

8. P received X $\rightarrow$ P sees X
9. P sees $(X_1, ..., X_n)$ 3 P sees X_i
10. $(P\ sees\ X_1 \wedge, ... \wedge P\ sees\ X_n) \;\rightarrow\; (P\ sees\ F(X_1, ..., X_n))$

Comprehending If a principal comprehends a message and sees a function of it, then understands that this is what it is seeing.

11. $P\ believes\ (P\ sees\ F(X)\ ) \;\rightarrow\; P\ believes\ (P\ sees\ X)$
12. $(P\ received\ F(X)\ \wedge\ P\ believes\ P\ seesX) \;\rightarrow\; P\ believes\ P\ received\ F(X)$

Saying A principal who has said a concatenated message has also said and sees the concatenates of that message. A principal who has recently said X has said X. A principal sees what he says.

13. $P\ said\ (X_i, ..., X_n) \;\rightarrow\; (P\ Said\ X_i\ \wedge\ P\ sees\ X_i)$
14. $P\ Says\ (X_i, ..., X_n) \;\rightarrow\; (P\ Said\ (X_i, ..., X_n)\ \wedge\ P\ Says\ X_i)$

Jurisdiction This is about to say that a particular party has full control over a message.

15. $(P\ controls\ \varphi\ \wedge\ P\ says\ \varphi) \;\rightarrow\; \varphi$

Freshness A concatenated message is fresh if one of its concatenates is fresh, and any effectively one-one function F (including encryption and decryption) of a fresh message is fresh.

16. $fresh(X_i) \rightarrow fresh(X_1, ..., X_n)$
17. $fresh(X_1, ..., X_n) \rightarrow fresh(F(X_1, ..., X_n))$

Nonce-Verification which says what ever message is said is said in current run of the protocol.

18. $(fresh(X) \wedge P\ Said\ X) \rightarrow P\ says\ x$

Symmetric goodness of shared keys A shared key is good for P and Q iff it is is good for Q and P.

19. $P \overset{K}{\longleftrightarrow} Q \equiv Q \overset{K}{\longleftrightarrow} P$

Having A principal has a key iff he sees it.

20. $P\ has\ K \equiv P\ sees\ K$

4.3 Syntactic Analysis

In this section we discuss about the syntactic protocol analysis technique in SVO logic which is given in [5], which is similar to the techniques given in BAN and AT. In SVO Syntactic Analysis the first step is *"protocol idealization"*. Consider a protocol step in which a key server S distributes a key to principal A for the purpose of talking with B. this can be written as,
$S \longrightarrow A: \{T_s B, K_{ab}\}_{K_{as}}$
Which tells that S is sending a session key K_{ab} to A to communicate with B. And it should represent that K_{ab} is good key communication between A and B. This can be achieved through protocol idealization. Using SVO the above message can be idealized as,
$S \longrightarrow A: \{T_s B, A \overset{K_{ab}}{\longleftrightarrow} B\}_{K_{as}}$

Next step in Syntactic Analysis is *"protocol annotation* " in which we write down corresponding formulae in the logic. Formulae generated by annotation as the premise set in proofs of protocol goals. To generate this set we first write down the initial assumptions, these premises are true before the start of the protocol. Which we call *"Initial Assumptions"* BAN.

With the premise set established we attempt to derive various goals concerning the protocol. A proof is a sequence of formulae in the logic. Each line is either a premise, an axiom, or derivable from preceding lines via modus ponens or necessitation.

Briefly the syntactic analysis contains the following steps‘:

1. Protocol Idealization
2. Write initial assumptions in premise set
3. Protocol Annotation
4. Derive Protocol goals

We follow the same steps in formalizing DNSSEC in next section.

Chapter 5

Extending SVO for DNSSEC

In the previous chapters we saw how logic frame works are used in specifying and verifying security protocols. It was shown that modal logic systems like BAN, AT, GNY, VO and SVO are widely used in the semantic analysis of protocols. In this chapter we extend SVO logic so as to formally represent and reason about "Origin Authentication" and "Integrity" in Domain Name Security System.

5.1 Basic Inference rules

1. F1 —Modus Ponens: A $\wedge$ A $\rightarrow B \vdash B$
2. F2 —Necessitation: $\vdash A \rightarrow P$ *believes* A

5.1.1 Extended Axioms of SVO

1. A1 —K (Distributive Axiom) :
   P *believes* $\varphi \wedge P$ *believes* $(\varphi \rightarrow \psi\ ) \rightarrow P$ *believes* ψ
2. A2 —T (Truth Axiom):
   P *believes* $\varphi \ \rightarrow \ \varphi$
3. A3 —4 (Positive Introspection):
   P *believe* $\varphi \ \rightarrow P$ *believes* P *belives* φ
4. A4 —5 (Negative Introspection):
   $P \neg$ *believe* $\varphi \ \rightarrow P$ *believes* $P \neg$ *belives* φ

5.1.2 Axioms For Reasoning about DNSSEC

1. $P\ believes\ \varphi\ \wedge\ P\ believes\ (\varphi \rightarrow \psi) \rightarrow P\ believes\ \psi$
2. $P\ believes\ \varphi\ \rightarrow\ P\ believes\ (P\ believes\ \varphi)$
3. $P \overset{K}{\longleftrightarrow} Q,\ \wedge\ R\ received\ \{X\}_K \rightarrow\ Q\ said\ X$
4. $PK_\sigma(Q, K)\ \wedge R\ received\ \{X\}_{K_{-1}} \rightarrow\ Q\ said\ X$
5. $((PK_\delta(P, K_p)) \wedge (PK_\delta(Q, K_q)))\ \rightarrow\ P\ \overset{K_{pq}}{\longleftrightarrow}\ Q$
6. $P\ received\ (X_1, X_2, ..X_n)\ \rightarrow\ P\ received\ X_i$
7. $(P\ received\ \{X\}_K$ P has $\tilde{\mathrm{K}}) \rightarrow\ P\ received\ X$
8. P received X $\rightarrow$ P sees X
9. P sees $(X_1, ..., X_n)\ \longrightarrow$ P sees X_i
10. $(P\ sees\ X_1 \wedge, ... \wedge P\ sees\ X_n)\ \rightarrow\ (P\ sees\ F(X_1, ..., X_n))$
11. $P\ believes\ (P\ sees\ F(X)\ )\ \rightarrow\ P\ believes\ (P\ sees\ X)$
12. $(P\ received\ F(X)\ \wedge\ P\ believes\ P\ seesX)\ \rightarrow\ P\ believes\ P\ received\ F(X)$
13. $P\ said\ (X_i, ..., X_n)\ \rightarrow\ (P\ Said\ X_i\ \wedge\ P\ sees\ X_i)$
14. $P\ Says\ (X_i, ..., X_n)\ \rightarrow\ (P\ Said\ (X_i, ..., X_n)\ \wedge\ P\ Says\ X_i)$
15. $(P\ controls\ \varphi\ \wedge\ P\ says\ \varphi)\ \rightarrow\ \varphi$
16. $fresh(X_i)\ \rightarrow\ fresh(X_1, ..., X_n)$
17. $fresh(X_1, ..., X_n)\ \rightarrow\ fresh(F(X_1, ..., X_n))$
18. $(fresh(X)\ \wedge\ P\ Said\ X)\ \rightarrow\ P\ says\ x$
19. $P\ \overset{K}{\longleftrightarrow}\ Q\ \equiv\ Q\ \overset{K}{\longleftrightarrow}\ P$
20. $P\ has\ K\ \equiv\ P\ sees\ K$

In DNSSE if signature of hash is valid (i.e decrypted value of hash is equivalent to the calculated hash at client end), then the received RRset is valid. This can be represented in Propositional logic as follows

21 $hash\ \longrightarrow\ RRSET$

22 $DS\ \longrightarrow\ DNSKEY$

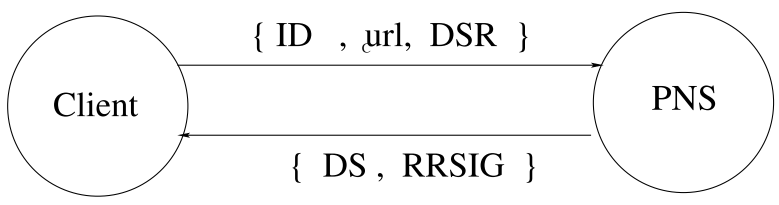


FIGURE 5.1: Client Requesting For DS Record Set

5.2 Formal Analysis Of DNSSEC

SVO logic system uses following four basic steps in formalizing analysis of protocols

1. Protocol Idealization
2. Write initial assumptions in premise set
3. Protocol Annotation
4. Derive Protocol goals

DNSSEC is developed to achieve security goals `Integrity` and `Origin Authentication`, we formalize DNSSEC to verify these goals,

Message 1: $Client \longrightarrow PNS : \{ID_C, url, ID_c\}$
Client sends a message to Parent zone Name Server for DS RRset of its zone.

Message 2: $PNS \longrightarrow Client : \{DS, RRSIG\}$
Client receives DS record set from Parent zone Name Server.

Objective 1: Origin Authentication:
The first security goal in DNSSEC is verifying that the message is received from the principal from whom it was originated and not altered by any intruders in between. To achieve this we need to derive premise "*Client believes* $DNSKEY_{ns}$." DNSSEC uses publickey signature to provide authentication. The client can verify the signature using public key i.e zone DNSKEY of the corresponding name server. Since anyone can create and publish public and private key pair, we need to verify the DNSKEY of the zone. this DNSKEY verification process is clearly explained in chapter 3. which establish authentication chain back to the previously learned DNSKEY.

Figure 5.2 represents the message transfers between Client and PNS. Here PNS is a principal who is parent to the Name server NS from which client received the DSN reply. In first message Client send a request message to the PNS for DS record set of the Name Server NS. As a reply PNS will send the DS record set to the client. And client will verify the $DNSKEY_{ns}$ using the DS record set received. The verification process using SVO is explained bellow.

We follow the Syntactic steps in SVO to verify DNSSEC Origin Authentication:

Step 1: Protocol Idealization

$PNS \longrightarrow Client : \{DS, RRSIG\}$ this message contain an encrypted message RRSIG, which can be rewritten in idealization as,

$PNS \longrightarrow Client : \{DS, \{hash\}_{DNSKEYpns^{-1}}\}$

Step 2: Write initial assumptions in premise set

To believe in zone's DNSKEY the client should believe in zone's parent DNSKEY, this can be an initial assumption in this, and client believes this is a public key to verify signature made by PNS. And Client believes the receives Messages are fresh.

A1. $Client\ believes\ DNSKEY_{pns}$

A2. $Client\ believes\ PK_{\sigma}(PNS, DNSKEY_{pns})$

A3. $PK_{\sigma}(PNS, DNSKEY_{pns})$

A4. $Client\ believes\ fresh(hash)$

Step 3: Protocol Annotation

In this step we write annotations on messages received, from Message 2 in protocol we can write the following annotation,

P1. $Client\ received\ \{DS, \{hash\}_{DNSKEYpns^{-1}}\}$

Step 4: Derive Protocol goals

We can derive the required goal $Clientt\ believes\ DNSKEY_{ns}$, as follows:
from P1 and Axiom 6 we can add following premise to the premise set.

D1. $Client\ receivedDS$

D2. $Client\ received\ \{hash\}_{DNSKEYpns^{-1}}$

From D1,D2 and 8

D3. $Client\ sees\ DS$

D4. $Client\ sees\ \{hash\}_{DNSKEYpns^{-1}}$

From D2 and A3 and Axiom 4 the following premise can be derived:

D5. $PNS\ said\ hash$

From A4, A5 and D3

D5. $PNS\ says\ hash$

D6. $PNS\ says\ DS$

From A4, N5, and 15

D7. hash

Form Necessitation Axiom and D7

Client believes hash

From Axiom 21 and D7 and from Modus ponens

D8. DS

From Axiom 22 and D8 and from Modus ponens

D9. $DNSKEY_{ns}$

From D9 and Necessitation

D10. **Client believes DNSKEY**$_{ns}$

With the deriveness of Client believes DNSKEY$_{ns}$, the first objective is achieved.

Objective 2: Integrity:

Message 1: $Client\ \longrightarrow\ NS\ :\ \{ID_C, url\}$

Message 2: $NS\ \longrightarrow\ Client\ :\ \{RRSET, RRSIG\}$

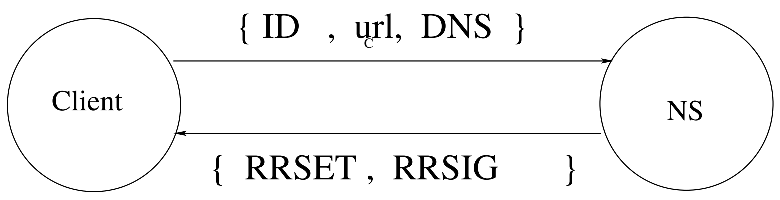


FIGURE 5.2: Client Requesting For Name-IP translation

To say DNSSEC achieves Integrity we need to prove that client is believing resource record sets received i.e " Client believes RRSET"

Step 1: Protocol Idealization

The message two in this can be idealized as

Message 2: $Client\ received\ \{DS, \{hash\}_{DNSKEYns^{-1}}\}$

Step 2: Write initial assumptions in premise set
Initial assumptions in this phase are

A1. Client believes DNSKEY$_{ns}$

A2. fresh(hash)

A3. $PK_\sigma(PNS, DNSKEY_{ns})$

A4. NS controls hash

Step 3: Protocol Annotation
Protocol annotations are

P1. Client received $\{RRSET,\ \{hash\}_{DNSKEYns^{-1}}\}$

Step 4: Derive Protocol goals
From P1 and 6

N1. Client received RRSET

N2. Client received $\{hash\}_{DNSKEYns^{-1}}$

From N1,N2 and 8

N3. $Client\ sees\ RRST$

N4. $Client\ sees\ \{hash\}_{DNSKEYns^{-1}}$

From N2 and A3 and Axiom 4 the following premise can be derived:

N5. $NS\ said\ hash$

From A2, N5 and 18

N6. NS says hash

From A4, N6, and 15

N7. hash

From Necessitation and N7

N8. Client believes hash

From N8, 21 and K-Axiom

N8. Client believes RRSET

By deriving N8. ***Client believes RRSET***. From this we can say that Integrity is achieved.

Chapter 6

Reasoning About DNSSEC (Implementation)

The reasoning about the correctness of DNSSEC protocol is implemented using C programming language. The system takes the details about the dns tree from a file which contains the information about each node in the network like name of the node, parent node name, name of the children and the zone to which it belongs to. In this implementation we are assuming that the parent of each node will be its zone server. And we considered each name server as recursive name server i.e the "If the desired naming resolution is not present at any server then server will forward that dns request to its parent. Example given bellow shows the dns tree we have used.

```
root$NULL$com$org$in$net

in$root$ernet$ac

ac$in

ernet$in$uohyd$jnu

net$root$whois$new

uohyd$ernet$mba$client$sche

client$uohyd

sche$uohyd

jnu$ernet

mba$uohyd
```

```
com$root$yahoo$google

org$root$mozilla$ieee

mozilla$org

ieee$org

yahoo$com

google$com

whois$net

new$net
```

Each row contains the information about name of the node, parent name and its children's names. By taking this information as input the program will generate dns tree.Through our program we have shown how the DNS resolution takes place from one node in the network to the other node in the network. In our example scenario we verified the correctness of DNSSEC protocol at the client "client.uohyd.ernet.in" for the dns resolution D.com. The program will generate the trace how the request is routed to the name server "com"

```
REQEST IS SENT TO : uohyd

REPLY SENT FROM uohyd TO client TO CONTACT ernet

REQUEST IS SENT TO : ernet

REPLY SENT FROM ernet TO client TO CONTACT in

REQUEST IS SENT TO : in

REPLY SENT FROM in TO client TO CONTACT root

REQUEST IS SENT TO : root

REPLY SENT FROM root TO client TO CONTACT com

REQUEST IS SENT TO : com
```

And the program will generate the rule base which contain the set of rules that being held by the client. The rules contains bel(believe) modality which we implemented in our program as a function "isbel" that verifies the belief as according to its possible world's. i.e if client believes any formulae then isbel can verify this by verifying the same formula at each node in the clients zone since zone of a client is its accessible possible world. For our example scenario this program will generate following rule base for the node "client"

I1- client bel DNSKEY $_{root}$

A1- client bel DNSKEY $\longrightarrow$ DNSKEY(axiom-T)

A2- client bel DNSKEY $\wedge$ clientrec $\{M\}$ DNSKEY $\longrightarrow$ client bel M(axiom-K)

D1- client sent Message:uohyd

D2- client rec Message:uohyd

D3- client sent Message:ernet

D4- client rec Message:ernet

D5- client sent Message:in

D6- client rec Message:in

D7- client sent Message:root

D8- client rec Message:root

D9- client sent Message:com

D10- client rec RRSET:com

D11- client sent Message:roo

D12- client rec Message:root

D13- client bel DNSKEY_com

D14- client bel RRSET:com

To say that DNSSEC is achieving the `Integrity` and `Origin Authentication` client need prove that it is believing received RRSET's and DNSKEY of the sender zone.the axioms D13 and D14 will tells that the required goal is achieved.

Chapter 7

Conclusion

Designing security protocols is a difficult task, because small flaws will open big gates for attackers to breach the system. So the security protocols need to undergo formal analysis to uncover the flaws before they are put in practice. This Project is concerned with developing a logic based framework so as to represent and reason about the DNSSEC (Domain Name System Security) protocol. DNSSEC provides security services to the existing DNS Protocol mainly through public key cryptography. But, it is well known that even the use of the most perfect cryptographic tools does not always ensure the desired security goals. This situation arises because of logical flaws in the design of protocols. In this thesis we formally specified and verified the DNSSEC protocol using modal logic so as to derive the correctness of the protocol. We used SVO logic (A variant of modal logic) to formalize DNSSEC. By representing and reasoning protocol messages of DNSSEC in SVO logic we showed a logical proof for `Origination Authentication` and `Integrity`. In order to automate the proof of correctness we implemented the logical framework in C programming language wherein our program will generate a premise set which contains set of rules that should hold true at the client side. We also outline a proof trace of how goals of protocols are derived from the set of premises.

Bibliography

[1] Michael Burrows, Martín Abadi, and Roger M. Needham. A logic of authentication. *ACM Trans. Comput. Syst.*, 8(1):18–36, 1990.

[2] Li Gong, Roger M. Needham, and Raphael Yahalom. Reasoning about belief in cryptographic protocols. In *IEEE Symposium on Security and Privacy*, pages 234–248, 1990.

[3] Martín Abadi and Mark R. Tuttle. A semantics for a logic of authentication (extended abstract). In *PODC*, pages 201–216, 1991.

[4] Paul C. van Oorschot. Extending cryptographic logics of belief to key agreement protocols. In *ACM Conference on Computer and Communications Security*, pages 232–243, 1993.

[5] Paul C. van Oorschot Syverson, P.F. On unifying some cryptographic protocol logics. In *Research in Security and Privacy, 1994.*, pages 14–28, 1994.

[6] Hui Liu and Minglu Li. Svo logic based formalisms of gsi protocols. In *PDCAT*, pages 744–747, 2004.

[7] Zhihong Huang Jin Jiang, Lei Li. Hidden-id authentication scheme and its svo-logic based formal analysis. *International Conference on Computational Intelligence and Security*, 8, February 2007.

[8] Guha R.K Muhammad S., Furqan Z. A logic-based verification framework for authentication protocols. volume 1, pages 49–80, March 2007.

[9] A.M. Mathuria R. Safavi-Naini P.R. Nickolas. On the automation of gny logic. In *Australasian Computer Science Conference*, pages 370–379, 1995.

[10] M. Larson D. Massey S. Rose R. Arends, R. Austein. Rfc 4033. 1, March 2005.

[11] M. Larson D. Massey S. Rose R. Arends, R. Austein. Rfc 4034. 1, March 2005.

[12] Geoff Huston. Dnssec - the theory. *The ISP Column*, 1, August 2006.

[13] Steven Cheung and Karl N. Levitt. A formal-specification based approach for protecting the domain name system. In *DSN*, pages 641–, 2000.

[14] Suranjith Ariyapperuma and Chris J. Mitchell. Security vulnerabilities in dns and dnssec. In *ARES*, pages 335–342, 2007.

Verifying the Correctness of Cryptographic Protocols Using "Convince"

Dr. Randall W. Lichota
Hughes Technical Services Company, P.O. Box 3310, Fullerton, CA 92834-3310
lichota@stars1.hanscom.af.mil

Dr. Grace L. Hammonds,
AGCS, Inc., 91 Montvale Avenue, Stoneham, MA 02180-3616
hammonds@dockmaster.ncsc.mil

Dr. Stephen H. Brackin
Arca Systems, Inc., 303 E. Yates St., Ithaca, NY 14850
Brackin@arca.com

Abstract[1]

This paper describes Convince, a tool being developed to facilitate the modeling and analysis of cryptographic protocols, particularly those supporting authentication. Convince uses a belief logic to facilitate the analysis and proof of desired properties of these protocols. Convince incorporates in its front-end a commercial computer-aided software engineering tool, StP/OMT, so that an analyst can model a protocol using a combination of familiar graphical and textual notations. Convince uses a Higher Order Logic theorem prover with automated support, so as to minimize the need for specialized theorem-proving knowledge. This paper describes how an analyst can use Convince to rapidly construct models of authentication protocols, and outlines a strategy for verifying their correctness. It discusses the integration of StP/OMT with the theorem-proving component and practical analysis techniques based on experience acquired through analyzing several published protocols.

1.0 Introduction

The growing importance of computer networks to information systems has highlighted the need for network security with high assurance. The work documented in this paper addresses one of many aspects of network security: authentication protocols. Modeling and verifying these protocols during the early phases of their software development can provide the assurance of correctness. This paper describes the Convince tool allowing an analyst to model and verify a protocol from within in a Computer Aided Software Engineering (CASE) tool environment, Software through Pictures™[2] Object Modeling Tool [IDE94b], and identifies general strategies to facilitate the verification of desirable protocol properties using a Higher Order Logic (HOL) theorem prover [GOR93].

An authentication protocol enables parties to a transaction to identify each other securely over a potentially unsecured network. Such protocols can also provide the basis for establishing that encryption keys are reliably distributed. By "verifying an authentication protocol", we mean identifying whether the protocol is susceptible to threats that include release of message contents, masquerading, unauthorized modification of messages, or replays.

To support authentication protocol verification, Convince incorporates an implementation of a "belief logic." [GON90]. Belief logic allows one to reason about the parties to a protocol in terms of what they can possess or believe about the messages exchanged at each step. Belief logic relies on the analyst's ability to define the

[1] The work described in this paper has been sponsored by Air Force Materiel Command's Electronic Systems Center/Software Center (ESC/AXS) Hanscom AFB, MA, and funded by Rome Laboratories, through contract numbers F19628-92-C-0006 and F19628-92-C-0008.

[2] StP is a trademark of the Interactive Development Environments.[IDE94a]

1063-9527/96 $5.00 © 1996 IEEE

goals of the protocol in that context, then provides a set of rules from which one can deduce whether the goals are met. Because the Convince verification process does not require assistance from the user, an analyst need only become skilled in the reasoning embodied by the rules of belief logic and their relation to proof failures. We have found that the verification process has much in common with debugging; a protocol model and its associated initial conditions and goals undergo incremental revision until the goals are proven or withdrawn.

Using Convince, we have analyzed a number of cryptographic protocols, including variations of the Kerberos authentication protocol [STE88], a public-key variant of Kerberos being proposed to support electronic commerce [CHU96, COX95], and the proprietary SPX protocol from Digital [BRA96c]. This paper shows an analysis of the Tatebayeshi-Matsuzaki-Newman (TMN) key distribution protocol, which has been used to illustrate the properties of other cryptographic protocol analysis systems.[KEM94]

This work was performed as part of the Portable, Reusable, Integrated Software Modules (PRISM) Project, at the Electronic Systems Center Software Center (ESC/AXS). Convince evolved from a task to identify advanced security technologies to support the Air Force mission. Many network security protocols have been proposed for the protection of "Unclassified but Sensitive" data. Unlike forms of verification used for the protection of classified data, Convince attempts to identify subtle problems in design, such as the potential for replays or masquerading. Prior to Convice, tools for analyzing protocol correctness did not fully incorporate belief-logic.

The remainder of this paper is organized as follows: Section 2 provides an overview of the Convince belief logic and discusses its evolution. Section 3 provides an overview of the Convince environment. Section 4 discusses how Convince can be used to automatically verify the goals of an authentication protocol, with an example. It presets guidelines for working through proof failures. Section 5 describes related work. Section 6 gives our conclusions and recommendations for future work.

2.0 Overview of Convince Belief Logic

2.1 Belief Logic Background

Belief logics had their origin in the logic for authentication developed by Burrows, Abadi, and Needham (known as BAN) for the verification of cryptographic protocols [BUR90]. In the BAN logic, an authentication protocol is transformed into a sequence of logical statements that are then subject to analysis. Gong, Needham, and Yahalom developed another logic, known as GNY, that was based on BAN but expressed at a lower level of abstraction [GON90]. Shortly thereafter, Gong identified the possibility of "infeasible" protocol steps, and added new constructs to eliminate that possibility [GON91]. The resulting logic was partially implemented, in HOL, at Odyssey Research Associates under the Romulus system [ORA94]. Using different HOL techniques, Brackin subsequently implemented a belief logic, called BGNY, implementing the full GNY logic with Gong's extension, and also extending this logic to address a wider variety of cryptographic protocols [BRA96a]. Convince implements the Brackin version of belief logic, referred to as "BGNY."

Through HOL constructs, BGNY provides significant extensions to GNY: it allows goals to be specified at different protocol steps, not just after the protocol has completed; it allows use of multiple algorithms for symmetric- and public-key encryption and hashing; it allows use of message authentication codes; it allows use of computed values (e.g., hashes) as keys; and it allows use of key-exchange functions.

The HOL BGNY theory contains the rules used in trying to prove that a protocol's initial conditions and messages indeed establish the protocol's goals. A list of informal versions of the main BGNY rules is given in an appendix to this paper.

2.2 Intermediate Specification Language

For the purpose of analysis, Convince expresses protocols in an Intermediate Specification Language (ISL) that is based partly on notation commonly used for describing cryptographic protocols and partly on BGNY belief-logic notions [BRA97]. It is designed to be close to natural language, yet still allow complicated expressions used in authentication protocols. Appendix 1 lists the main syntactic and semantic elements of ISL.

3.0 Convince Toolset

The Convince toolset takes advantage of commercially available, public-domain software for the most part, incorporating tool-specific programming to generate the outputs required under BGNY.

Three main tools, or software applications, make up Convince. These three provide a graphical object-oriented interface for describing the protocol; a facility for translating protocol descriptions into mathematical

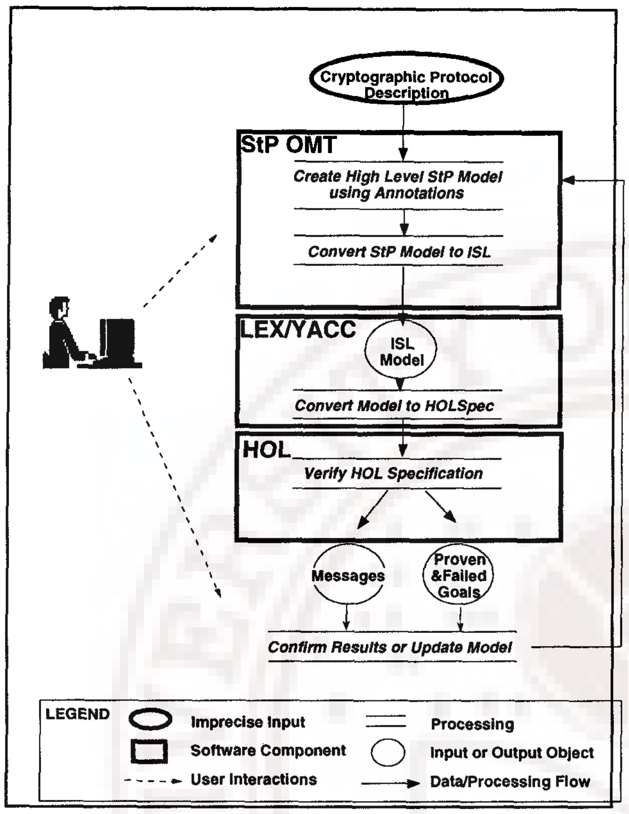


Figure 1. Process and Data Flow

specifications; and a formal theorem prover.[3] Each of these tools is augmented with scripts, macros, or other programming, as described in the sections that follow. Figure 1, "Process and Data Flow", shows the data flow between applications, and corresponding user actions and processes.

3.1 StP/OMT

The Software through Pictures™ (StP) Object Modeling Tool (OMT) is a Computer-Aided Software Engineering (CASE) tool that facilitates the development of software systems via the Object Modeling Technique (OMT) methodology[4] [IDE94b]. StP/OMT provides a set of graphical editors that can be used to create OMT software models. Convince utilizes the StP/OMT graphical editors for the following software models: Use Cases, Event Traces, and Dynamic Models. It is here that a user has the most interaction, creating and modifying protocol diagrams and descriptions. Scripts under OMT automatically generate ISL specifications.

3.2 LEX/YACC

LEX and YACC are UNIX software development tools [DYS96] employed in Convince to parse the ISL protocol specifications and generate the form required for HOL processing. The parser/generator process is completely automated for the convinence of the user, to simplify formal specification development.

3.3 HOL

The HOL 90.7 theorem prover is the main verification component underlying Convince. By design, Convince hides the details of HOL from the user. From a user's point of view, the HOL component just accepts HOL versions of protocols and their desired properties, and returns whether the goals have been deduced from statements in these specifications.

The steps shown in Figure 1 are described in detail below.

4.0 Verification Process under Convince

4.1 Example: TMN Protocol

TMN is an example of a *flawed* protocol; numerous security problems could arise if it is implemented as specified. We use this example to show how an analysis of such a protocol would be undertaken.

The TMN Protocol was originally proposed to distribute session keys among ground stations in a mobile communication environment. All keys are obtained with the cooperation of a Server station, for which all parties have the public key of the public/private key pair.

In this example, A and B are two arbitrary ground stations and S is the Server. These three are the only principals, or parties, to the protocol. We also show rsa as the public-key encryption algorithm, Ra and Rb as random numbers, PKs as the Server's public key, and M as the actual message.

The basic steps of the protocol are shown below. See Appendix 1 for the complete description of the ISL notation.

[3] The translation/verification subsystem of Convince is also known as the Automatic Authentication Protocol Analyzer (AAPA). The AAPA can accept ISL specification for verification regardless of how they were produced.

[4] OMT was originally developed by James Rumbaugh and his associates at GE in the late 1980's. More recently, the methodology has been enhanced through the efforts of Jacobson, Booch, and Rumbaugh.

```
1. A -> S: {Ra}rsa(PKs), A, B;
2. S -> B: A;
3. B -> S: {Rb}rsa(PKs);
4. S -> A: {Rb}e(Ra)||(SharedSecret A B
Rb);
5. A -> B: {M}des(Rb);
```

In step 1, principal A signals the Server to send a key for itself and B, by sending a random number encrypted under the Server's public key, PKs, that can only be read by the Server. In step 2, the Server forwards the request to B. In step 3, B returns to the Server a second random number, Rb, to be used as a key between itself and A. In step 4, the server encrypts (using algorithm 'e'[5]) the key from B using the random number it received from A. In step 5, A uses the key Rb to encrypt a message M for B.

4.2 Creating the High Level StP Model with Target Protocol Goals

A Convince user enters the TMN protocol into StP using the three formats: a **use case** diagram; an **event trace** diagram; and a **dynamic model**.

An StP use case model for TMN is shown in Figure 2. "Use Case Diagram for TMN." Use case models are extensions to the original OMT formalism that were developed by Jacobson. We have adapted this notation to provide a structure for organizing protocol specifications. A use case consists of one or more elements known as *actors* connected to one or more processes. In Convince, actors represent principals in an authentication protocol, while processes correspond to event traces representing specific protocol scenarios. These diagrams define a family of protocols that share common elements (e.g., principals, encryption algorithms, certificate authorities).

In Figure 2, "Use Case Diagram for TMN," the window to the left allows an analyst to choose from among a number of different use case models. The window to the right shows the model selected.

Event trace diagrams[6] are used to describe the sequence of message transfers that comprise an authentication protocol. These diagrams are the main source of input to generate the ISL specification. As shown in Figure 3, an event trace consists of a context object, a set of object classes, and a set of directed line segments that denote message transfers. A context object is represented as an ellipse enclosing the name of the protocol (taken from the process element in the associated use case diagram) and scenario to which the event trace is linked. The context object is also associated with data-element definitions needed to automate the analysis. Each object class corresponds to a vertical line segment labeled with the name of a principal of the authentication protocol. Each message transfer is labeled with a text string denoting the nature of the message (e.g., request for key) and the stage of the protocol when the transfer occurs.

In Figure 3, "Event Trace Model for TMN," the upper window shows the first four protocol steps as identifiers. The lower window shows, as an OMT *annotation*, the details of the currently highlighted identifier.

Dynamic models (not shown) are used to show the states of each principal after each protocol step.

In order to completely describe and verify the correctness of authentication protocols, we had to extend the notation provided by OMT. In Convince, we did this primarily by using annotations. Annotations associated with the context object within an event trace identify the names of principals, keys, cryptographic functions, hash functions, and other functions referenced in other annotations (e.g., arithmetic functions). Annotations also record data type definitions, initial conditions with any assumptions about all possessions and beliefs held by the principals, and protocol goals. For each invertible function, an annotation relates the function to its inverse, possibly through the use of encryption/decryption keys.

The complete ISL for TMN is shown in Figure 4, "TMN Specification in ISL."

4.3 Converting the StP Model to ISL

The analyst directs Convince to verify a model of an authentication protocol through a command option. Using this option, OMT makes the initial conversion to an ISL specification. The translation is implemented by an StP Query Reporting Language (QRL) script that extracts relevant elements of the model from an StP repository and assembles them to form an ISL file. This file conforms to a textual language whose syntax is a superset of the annotation syntax employed in Convince. The analyst can immediately start the verification process after translation. The analyst can also choose to review and/or edit the ISL directly from an OMT-invoked text editor.

At this point, Convince can detect syntax errors introduced by the user in annotations (e.g., errors in punctuation or missing declarations).

4.4 Converting the ISL Model into a HOL Specification

[5] The TMN protocol calls for 'e' to be chosen such that it is commutative, that is, that Ra encrypted by Rb provides the same result as Rb encrypted by Ra. This particular restriction has been shown by Kemmerer to cause a flaw in TMN [KEM94]. This flaw is *not* detected by BGNY and hence Convince; BGNY does not recognize "commutative" as a meaningful attribute.

[6] In cases where multiple scenarios are to be modeled, a process may be associated with multiple event trace diagrams.

To verify a model, Convince must first translate it into HOL. Convince provides essentially transparent translation from ISL to HOL. We added this and subsequent steps specifically to remove the burden from users of dealing with HOL directly. Even the error messages produced by Convince during the verification stages use ISL formats.

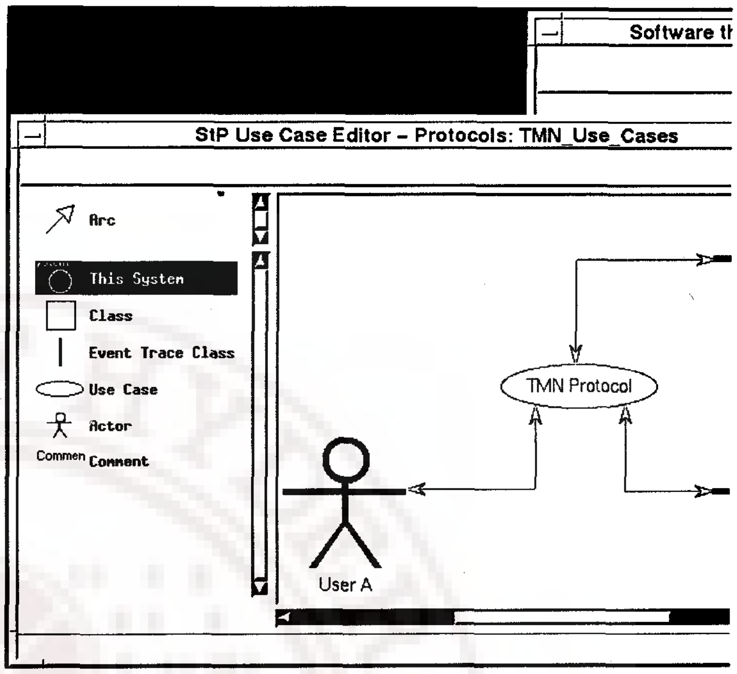
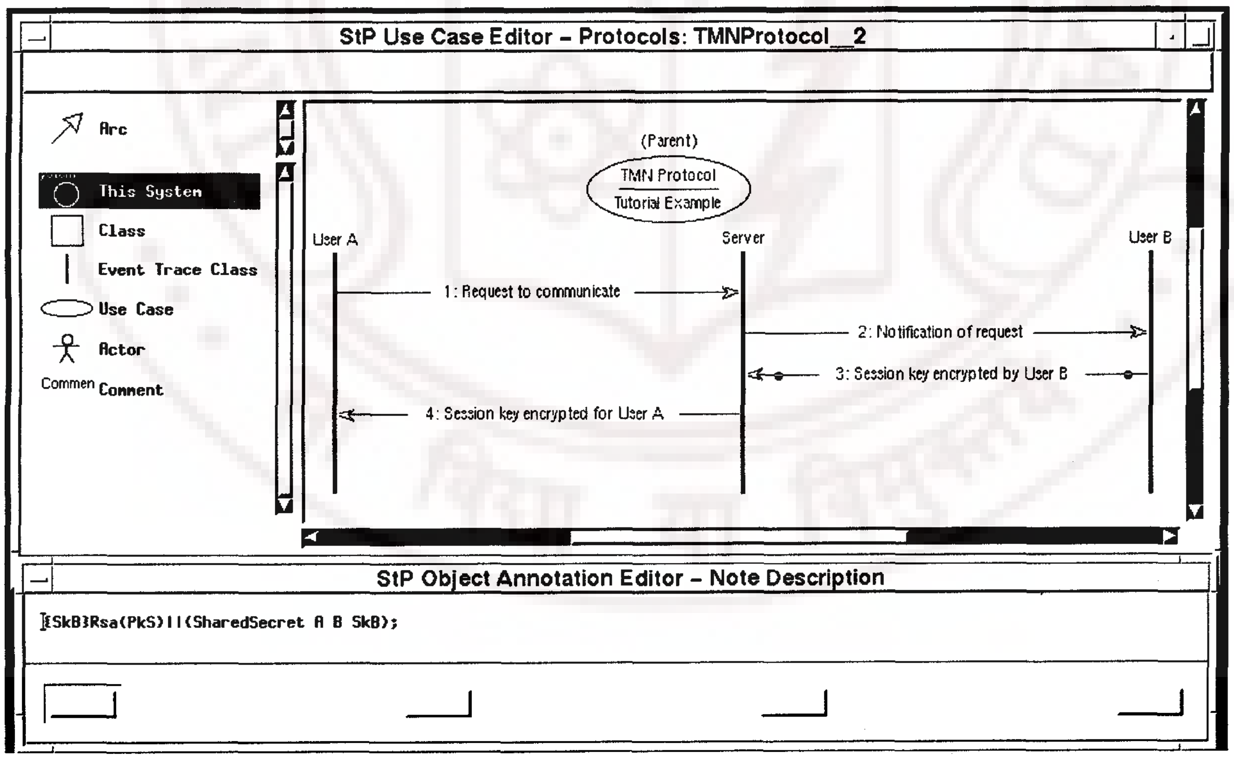


Figure 2. Use Case Diag


StP Use Case Editor - Protocols: TMNProtocol__2
Arc
This System
Class
Event Trace Class
Use Case
Actor
Commen Comment
(Parent)
TMN Protocol
Tutorial Example
User A
Server
User B
1: Request to communicate
2: Notification of request
3: Session key encrypted by User B
4: Session key encrypted for User A
StP Object Annotation Editor - Note Description

```
{SkB}Rsa(PkS)||(SharedSecret A B SkB);
```

Figure 3. Event Trace Model for TMN

While this translation step implements syntactic checking, it also performs the first semantic checking. Semantic errors generally fall into one of the following categories: type errors; errors relating to parameter substitution; goal stage numbering errors; missing or extraneous source designations; and incorrect formulations of conveyance goals.

Type errors can arise for a variety of reasons, including improper usage (e.g., specifying an encryption function where the context requires a hash function), naming mistakes, and failing to define a data element in the DEFINITIONS section of the model. Errors relating to parameter substitution occur when an incorrect number of arguments is supplied to a function (e.g., providing less than 2 arguments to a key-exchange function or supplying a key with a "keyless" hash function), or using arguments of improper type. Convince generates goal stages automatically from a model, so they usually will not be a source of error, but they can be wrong if an analyst directly edits an ISL specification file.

4.5 Verifying HOL Specifications

In the following descriptions, comments on the TMN proof are shown in *italics*.

4.5.1 Reviewing Assumptions and Goals. When constructing a model of the protocol, the analyst describes it abstractly, with careful attention to the assumptions (identified as initial conditions in ISL), the goals, and the assertions associated with message transfers. Convince performs model verification by attempting to prove that the goal statements of a model logically follow from its initial conditions, the message transfers at the protocol's different stages, and the BGNY rules. These initial conditions will depend on what the analyst expects to be true for each principal prior to the start of or during the protocol. Conditions include beliefs about properties of message items, such as "the message is fresh" or "the message is recognizable".

For the most part, goal statements should clearly derive from the original cryptographic protocol description used to construct the protocol model. Knowledge of the rules of the BGNY logic is useful in helping to identify which types of beliefs are significant in establishing the properties expressed in the goal statements. Through this knowledge, an analyst can compile a concise set of rules covering those conditions needed for a successful proof. In all cases, however, an analyst should take care to include only those initial conditions and goals that could reasonably be ascribed to the protocol principals.

```
/* TMN KEY DISTRIBUTION PROTOCOL */

DEFINITIONS:

PRINCIPALS: A, B, S:
SYMMETRIC KEYS: Ra, Rb;
PUBLIC KEYS: PKs;
PRIVATE KEYS: ^PKs;
OTHER: M;
ENCRYPT FUNCTIONS: e, rsa, des;

e WITH ANYKEY HASINVERSE e WITH ANYKEY:
des WITH ANYKEY HASINVERSE des WITH ANYKEY;
rsa WITH ^PKs HASINVERSE rsa WITH PKs;
rsa WITH PKs HASINVERSE rsa WITH ^PKs;

INITIALCONDITIONS:
A Received Ra, M;
A Received A, B, PKs, des, rsa, e;
A Believes
   (A Recognizes A; A Recognizes B;
    PublicKey S rsa PKs
   );
B Received Rb;
B Received B, S, PKs, des, rsa;
B Believes
   (B Recognizes A; B Recognizes B;
    Trustworthy S;
    PublicKey S rsa PKs
   );
S Received S, A, B, des, rsa, e, PKs, ^PKs;
S Believes
   (PublicKey S rsa PKs;
    PrivateKey S rsa ^PKs
   );

PROTOCOL:
1. A -> S: {Ra}rsa(PKs), A, B;
2. S -> B: A;
3. B -> S: {Rb}rsa(PKs);
4. S -> A: {Rb}e(ra)||(SharedSecret A B
Rb);
5. A -> B: {M}des(Rb);

GOALS:
1. S Possesses Ra;
3. S Possesses Rb;
   S Believes (B Possesses Rb);
4. A Believes (SharedSecret A B Rb);
5. B Possesses M;
   B Believes (A Possesses M);
```

Figure 4. TMN Specification in ISL

The initial specification of TMN, shown in Figure 4, is typical in having more lines of initial conditions than of protocol. However, they were derived fairly simply by *examining what had to be true in order for the protocol to execute. The goal section, on the other hand, is fairly short, generally identifying that certain data items should be in the possession of the principals at different steps of the protocol, and that certain beliefs should be held as a result. The problem the analyst faces is trying to prove the goals with the information given.*

4.5.2 Proving the Goals. Once Convince successfully translates a model into HOL, it invokes utilities to automatically prove the *user-specified* goals expressed in the model. This is done by establishing them as consequences of *default* goals that Convince generates automatically and tries to prove [BRA96b]. For each message sent, the proof utilities attempt to prove the following default goals:

- the message sender is capable of sending it;
- the message recipient can decrypt each encrypted portion of the message; and
- for each encrypted or hashed portion of the message, the message recipient
  - can confirm the true originator of that portion,
  - has adequate reason to believe the properties the protocol assumes of that portion, and
  - has adequate reason to believe that the portion's originator also believes these properties.

While many default goals are false, the true default goals typically have the goals expressed in the model as easy consequences -- if the latter are indeed true.

In the TMN specification, in the first step, the goal is for the Server S to receive the data (Ra) from A. Under the BGNY rules, the reception occurs if A has the items needed to send the message. Since A possesses all the pieces to effect the transfer—Ra, rsa, and PK—the reception goal can be proved. This particular goal is modest, because we have not proved that the Server can believe that A sent the message.

4.5.3 Generating and Proving Default Goals. For each protocol step, or stage, Convince first attempts to prove the set of default goals. Failure to prove a default goal does not necessarily denote a problem because the goal may become true at a later stage.

Once Convince has finished attempting to prove the default goals for the current stage of the protocol, it attempts to establish that the user-specified goals expressed at that stage in the model follow from the proven default goals. If it cannot do this, Convince displays the goal in the model that it could not prove, prints a brief statement describing the reason for the failure, and terminates the execution of its proof process.

Convince terminates a proof when it has successfully proven all the goal statements in the protocol model, when it is unable to prove one of the user goals, or when it detects a feasibility failure (explained below). In attempting to prove default goals, Convince will retry any failed goals associated with previous stages if they might have since become true; it retries goals associated with the current stage only if it finds a goal from an earlier stage that has become true. Convince also terminates execution if a stage of the protocol is infeasible, that is, if the proof utilities determine that the protocol specification is incomplete. Otherwise, Convince moves on to the next stage of the protocol and attempts to prove its corresponding goals. When Convince has successfully proven the goals for all of the stages of the protocol, it completes with a message to this effect.

In TMN, stage 2: `S -> B: A,` *no user goals are specified, because the analyst realizes that there are no assurances that data in the clear will not be tampered with.*

In stage 3, B returns `{Rb}rsa(PKs) to S,` *where Rb is expected to be used as a session key with A. Here, the Server legitimately should want to believe that Rb is actually held by B. (Indeed, this is true for the A at stage 1, but that goal was not included.) However, since anyone with the Server's public key could generate such a message, S has no basis for believing the random number came from B. Even if S believed only B (and A) had its public key (a new "shared secret" initial condition), the BGNY logic requires that S would also need to believe the following about the message:*

1) *the message is fresh (recently produced) -- e.g., a current timestamp, added to the message, would make it possible to believe the message is fresh.*
2) *the message is recognizable (after being decrypted) -- e.g., a name added to the message might make the whole message recognizable.*

Without the freshness attributes on the message, it could be possible to accept an intercepted and replayed message as if it were legitimate. Without the recognizable attribute on the message, anyone could send anything to the Server, and the data could be interpreted as a legitimate request. Consequently, Convince would automatically add these two conditions as default goals to be proved.

4.5.4 When Goals Fail to Prove. Convince's failure to prove goals, either user-specified or default, in a protocol model can be due to a variety of causes. A feasibility failure occurs for a specific protocol stage when one or more data items transferred between principals has not been identified as being in the possession of the sending principal prior to that stage. This is usually due to one of the following conditions:

1) the data items were not listed in the initial conditions as being received, or properties of the data were not shown as believed by the sender;
2) statements associated with data items listed in the initial conditions are missing from the formula containing these data items at a subsequent protocol stage.

If the problem was due to simple oversight on the part of an analyst, it can sometimes be corrected by simply augmenting the initial conditions to include additional statements of reception or creation of the data items. In other instances, the problem can be corrected by modifying the data transfers defined in previous stages. In cases where the proof utilities determine that the protocol specification is incomplete, the problem is usually a missing inverse definition for an encryption function.

In still other cases, the problem is a symptom of a real flaw in the protocol that could necessitate significant changes to the protocol itself.

In the TMN example, it is not possible to prove default goals on freshness and recognizability because the data sent to S, Rb, was not identified in the initial conditions as having those properties, and indeed, it is not necessarily true that Rb would ever be recognizable. To correct this problem, as well as the "lack of freshness" problem, one or two items with those properties might be added to the message -- but this means that the protocol has been changed. The assumption that only A and B have B's public key would also need to be added.

The failure to prove a goal statement can occur for a variety of reasons. Thus the first step towards resolving the failure is to identify its cause. Many times this can be done by identifying which default goals Convince proved and which it failed to prove for the stages up to the one at which the failure occurred. To obtain this information, the analysis can select the Show Failed Goals command option. This causes Convince to display the results of the proof attempts, expressed in ISL. This information is organized by protocol stage, starting with the most recent one, and lists the failed default goals, the subgoals proven to be true, and the subgoals needed to complete the proofs but not yet proven to be true. The latter usually show the source of the problem.

The outcome of a successful proof should not be viewed as merely "true" or "false". A successful proof must be tested for realism. That is, the assumptions, or initial conditions, should be examined to see whether they are valid, and conclusions should be examined to see if they are necessary and sufficient for the security properties desired from the protocol. An unsuccessful proof can indicate that a goal is not provable, but it can also indicate that the goal is overly constraining.

Knowledge of the proof rules can be important in determining the cause of a goal failure. A description of the main rules is listed in Appendix 2.

In TMN, the addition of fresh and recognizable data both to steps 1 and 3 would only cover the Server's "comfort" about the origins of certain data. The Server subsequently would have to convey to A its belief that the data is a secret key to A. Here again, more data (and corresponding initial conditions) would have to be added at step 4 to address the missing attributes. For the same reason, similar data would be needed at step 5.

5.0 Related Work

An early effort to automate the analysis of authentication protocols was embodied in the Romulus prototype [ORA94]. In Romulus, belief logic was implemented using HOL to form of a theory of authentication (crypto_90). This tool required a user to express all initial conditions, protocol stages , and goals as HOL statements. The verification process involved directly applying HOL proof tactics using the rules defined in crypto_90. Romulus depends heavily on the HOL environment, requiring that a user be thoroughly familiar with HOL. In addition, the implementation of crypto_90 covered only one encryption algorithm, and only symmetric-key encryption, not public-key encryption.

Other tools include the NRL Protocol Analyzer, MITRE's Interrogator, the EEC tool within Abrial's B machinery. The NRL Protocol Analyzer is a rewrite system that determines whether an undesirable state can be reached from a set of initial states [MEA91]. An example use of this tool would be determining whether any amount of manipulation of exposed data by an intruder could produce a secret key. The NRL Protocol Analyzer differs from Convince in the type of goals it can handle. Specifically, the NRL Protocol Analyzer attempts to show the absence of all possible flaws in a protocol, while Convince attempts to establish that certain beliefs are justified. The NRL Protocol Analyzer is also a low level tool that requires a user to define the undesirable states. It does not utilize belief logic for de fining initial conditions or protocol goals.

The Interrogator, which was built by Millen, generates a large number of paths through the protocol ending in an insecure state, and sees whether any of these begin in a valid initial state [MIL84]. It does not prove that there are no such paths, and finds similar flaws as the NRL Protocol Analyzer.

The EEC tool within Abrial's B machinery is rich enough to express a variety of conditions, but is primitive compared to Convince.

Recently Tom Schubert and his students at Portland State University created an embedding of the SVO logic within the HOL theorem proving system [SCH95]. The SVO logic represents an effort to harmonize different variations of belief logic into a single unified formalism. They used the resulting tool to specify and verify the correctness of key-escrow protocols. As a byproduct of this work, they created an infrastructure to support interactive proofs in HOL.

6.0 Conclusions and Recommendations

This paper has described a software environment, Convince, that makes it possible to perform careful formal analyses of authentication protocols quickly and easily. It described the principal components of this environment, along with an approach for constructing models of authentication protocols. It noted that the verification facilities provided by Convince are highly automated, requiring little user interaction to prove whether desired properties of a protocol hold. However, because these properties must be expressed in a form of belief logic, knowledge of the latter's underlying rules is essential for effective use of Convince.

Although Convince uses the HOL 90.7 theorem prover to implement its verification facilities, statements of initial conditions and goals are expressed in ISL. This allows the user/analyst to focus primarily on those elements of importance to the domain of authentication protocols, while minimizing the need to become familiar with proof techniques for the underlying logic. In our experience using Convince, we have found that the tool provides a significantly greater degree of automation than previous tools based on belief logic. However, we have found that the efficiency of the verification process depends on the strategies used to debug ISL specifications. In these instances, we have found the Convince capability to display both proved and failed goals essential for successful debugging.

In using Convince, we typically create an initial model using StP/OMT, translate it to an ISL specification, attempt to prove the desired properties, iteratively modify the specification, and then update the model to reflect the changes. Where the desired properties cannot be established within a few iterations, we have had to revisit the original "informal" specification of the protocol.. Based on our understanding of belief logic, we have been able to postulate modifications to the protocols that would serve to mitigate some potential protocol vulnerabilities

To date Convince has been used to model and analyze several authentication protocols that incorporate a variety of cryptographic elements (e.g., symmetric- and public-key encryption, digital signatures, and key exchanges). We are continuing to analyze new protocols with Convince. We are also continuing to enhance the user interface to simplify model creation and proof for the protocol analyst.

References

[BRA96a] S. H. Brackin, "A HOL Extension of GNY for Automatically Analyzing Cryptographic Protocols." *Proceedings of the 9th IEEE Computer Security Foundations Workshop*, 1996.

[BRA96b] S. H. Brackin, "Deciding Cryptographic Protocol Adequacy with HOL: The Implementation." *Proceedings of the 1996 International Conference on Theorem Proving in Higher Order Logics*, 1996.

[BRA96c] S. H. Brackin. "Automatic Formal Analyses of Cryptographic Protocols." *Proceedings of the 19th National Information Systems Security Conference*, 1996.

[BRA97] S. H. Brackin, "An Interface Specification Language for Automatically Analyzing Cryptographic Protocols." To appear in the *Internet Society Symposium on Network and Distributed System Security*, San Diego, CA, February, 1997.

[BUR90] M. Burrows, M. Abadi, and R. Needham, "A Logic of Authentication." *ACM Transactions on Computer Systems*, 8(1), February 1990.

[CHU96] J. Chuang, Public-Key Based Ticket Granting Services in Kerberos, Internet Draft RFC 1510 (valid through September 1996).

[COX95] Benjamin Cox, J. D. Tygar, Marvin Sirbu, "NetBill Security and Transaction Protocol." *Proceedings of the First Workshop on Electronic Commerce*, 1995.

[DYS96] P. Dyson, *The UNIX Desk Reference.* Sybex, Inc.: San Francisco, CA: 1996.

[GON90] L. Gong, R. Needham, and R. Yahalom, "Reasoning about Belief in Cryptographic Protocols." *Proceedings of the 11th IEEE Symposium on Research in Security and Privacy*, 1990, pp. 234-248.

[GON91] L. Gong, "Handling Infeasible Specifications of Cryptographic Protocols." *Proceedings of the Computer Security Foundations Workshop IV*, 1991, pp. 99-102.

[GOR93] M. Gordon and T. Melham, *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*. Cambridge University Press, Cambridge, UK, 1993.

[IDE94a] Interactive Development Environments, *Fundamentals of StP. Rel. 1*, February 1994.

[IDE94b] Interactive Development Environments, *Creating OMT Model. Rel. 1*, February 1994.

[KEM94] R. Kemmerer, C. Meadows, J. Millen, "Three Systems for Cryptographic Protocol Analysis." *Journal of Cryptology*, Vol 7, 1994, pp. 79-130.

[MEA91] C. Meadows, "A System for the Specification and Analysis of Key Management Protocols." *Proceedings of the Symposium on Security and Privacy*, 1991, pp. 182-147.

[MIL84] J. Millen, "The Interrogator: A Tool for Cryptographic Protocol Analysis." *Proceedings of the Symposium on Security and Privacy*. 1984, pp. 134-141.

[ORA94] Odyssey Research Associates, *Romulus User's Manual*, March 1994.

[SCH95] T. Schubert and S. Mocas, "A Mechanized Logic for Secure Key Escrow Protocol Verification." *1995 International Workshop on Higher Order Logic and its Applications*.

[STE88] J.G. Steiner, C. Newuman, and J. I. Schiller, "Kerberos: An Authentication Service for Open Network Systems." *Proceedings of the USENIX Winter Conference*, 1988, pp. 191-202.

Appendix 1. Symbolic Notation in BGNY/ISL

NOUNS

P, Q Principals, or parties to the protocol. If used in a message, these denote names.

x Data item or message containing data items.

s Statement.

k Key.

alg Algorithm for encryption, signature, or hash (such as a message digest).

NOTATION

x || *s* Often the fact that a principal sends a data item x reflects the principal's current state of mind. A statement s concerning this state can be attached to the data item to denote the meaning of sending the data item.

x,y Elements of a list.

(s1;s2) List of statements.

H(x) Function, normally used to identify a hash (or digest) of *x* , as with the MD5 algorithm.

{ x } alg(k) Encrypt content *x* with key *k* using algorithm *alg*, e.g., RSA or DES.

{ H(x) } alg(k) Sign *x* by hashing *x* and encrypting the result using algorithm *alg* and key *k*.

[x] (H, alg)(k) Sign *x* by hashing *x* and encrypting the result with algorithm *alg* and key *k*, then append the result to *x* in the clear.

P -> Q: x *P* sends message *x* to *Q*

STATEMENTS

P Receives x. Principal *P* receives data *x*, possibly after performing some computation such as decryption.

P Possesses x. *P* is able to put *x* into future messages in the current session. *P* is considered to possesses a data item that *P* can compute from the data items *P* already holds.

P Conveyed x. *P* originated, or created, and subsequently transmitted *x* to another principal

Fresh x. Data item *x* has never been used in a previous run of the protocol. An example is a "nonce" -- a random number generated for the purpose of being fresh

P Recognizes x. *P* can identify *x* as meaningful information of an expected form.

P Believes s *P*. believes or is entitled to believe that statement *s* holds.

Trustworthy P. If *P* was the source of a data item with an associated statement, this is adequate reason for believing that the associated statement is true.

SharedSecret P Q x. *P* and *Q* may properly use *x* to prove each other's identity. They may also use it as (or derive from it) an encryption key for secure communication.

PrivateKey Q alg k. The key *k* used in conjunction with the algorithm *alg* is one of *Q*'s private keys.

PublicKey Q alg k. The key *k* used in conjunction with the algorithm *alg* is one of *Q*'s public keys.

Appendix 2. Example BGNY Rules

The following are descriptions of the main rules used by the Convince proof process to show that protocols have their desired properties. The rules are actually implemented in HOL [BRA96a].

RULES ON POSSESSION

P1 & P2: Received items are possessed, and pairs of things that are possessed are also possessed.

P4: If functions, keys, and data items are possessed, then the results of applying these functions with these keys to these data items are possessed. Further, if a computed result is encrypted or a hash value, and a particular statement is believed, then the result can be sent in messages, even if the protocol assumes that the result is not sent unless this statement is believed.

P5: If a pair is possessed, then both elements of this pair are possessed.

P7: If the inverse of a function is possessed, and a value computed by applying this function to a data item is possessed, then the data item is possessed. This rule includes getting possession of plaintext from possession of ciphertext.

P8: If two expressions denote the same key-exchange value, then if one of them is possessed the other is also possessed.

RULES ON RECEIVING

R2: If a pair is received, then both elements of this pair are received.

R3: If a function's inverse is possessed, and a value computed by applying this function to a data item is received, then the data item is received. This rule includes getting reception of plaintext from reception of ciphertext. .

RULE ON TRUSTWORTHINESS

T1: If a principal P believes that something was conveyed by a principal Q that P trusts, P believes that this data was created for the current run of the protocol, and the protocol assumes that this thing would not have been sent unless Q believes a statement s, then P can believe s and believe that Q believes s.

RULES ON CONVEYANCE

BC1: If a principal P receives something that can be decrypted to something meaningful with a principal Q's public key, it must have been encrypted with Q's private key, so P can believe it originally came from Q.

BC2: If a principal P receives something that can be decrypted to something meaningful with a symmetric key P shares only with Q, if P can identify this thing as created for the current run of the protocol, and if this thing was not contained in something P sent out earlier in the current run of the protocol, then P can believe this thing originally came from Q. (The proof tools check the ``not contained in something P sent out earlier" condition automatically.)

BC3: If a principal P receives a hash code that is either the hash of something P shares only with Q, or is a key-dependent hash of something P possesses using a key that P shares only with Q, if P can identify this hash code as created for the current run of the protocol, and if this hash code was not contained in something P sent out earlier in the current run of the protocol, then P can believe this hash code originally came from Q. (The proof tools check the ``not contained in something P sent out earlier" condition automatically.)

BC4: If a principal P receives something that can be decrypted to something meaningful with P's private key, if its decryption contains something that P shares only with Q, if P can identify this thing as created for the current run of the protocol, and if the received thing was not contained in something P sent out earlier in the current run of the protocol, then P can believe it originated with Q.

BC5: If a principal P receives something that can be decrypted to something meaningful with a symmetric key P shares only with Q, and if P can identify this thing as not contained in anything P sent out at any earlier time, then P can believe this thing originally came from Q.

BC6: If a principal P receives a hash code that is either the hash of something P shares only with Q, or is a key-dependent hash of something P possesses using a key that P shares only with Q, and if P can identify this hash code as not contained in anything P sent out at any earlier time, then P can believe this hash code originally came from Q.

BC7: If a principal P receives something that can be decrypted to something meaningful with P's private key, if its decryption contains something that P shares only with Q, if P can identify this thing as created for the current run of the protocol, and if the received thing was not contained in something P sent out earlier in the current run of the protocol, then P can believe it originated with Q.

RULES ON FRESHNESS

BF1 & BF2: If something can be identified as having been created for the current run of the protocol, then anything containing or computed from this thing can be identified as having been created for the current run of the protocol.

BF5: If two expressions denote the same key-exchange value, then if one of them is can be believed to have been created for the current run of the protocol then the other can be also.

BP2: If a principal can be believed to have originated something created for the current run of the protocol, this principal can be believed to still possess it.

RULES ON RECOGNIZABILITY

BR1 & BR2: If something can be identified as meaningful information (or be recognized), then anything containing or computed from it can be identified as meaningful information

RULES ON SHARED SECRETS

BS1: The “is a shared secret” relation is symmetric.

BS3: A key generated by a key-exchange function from one principal's private key and another principal's public key can be believed to be a shared secret for these principals; the same is true for key-exchange functions that also take additional arguments.

BS4: If two expressions denote the same key-exchange value, then if one of them is a shared secret the other is also a shared secret.

C3PO: a Tool for Automatic Sound Cryptographic Protocol Analysis

Anthony H. Dekker
Defence Science and Technology Organisation and
Department of Computer Science, Australian National University
Postal: PO Box 3925, Manuka ACT 2603, Australia
E-mail: dekker@acm.org

Abstract

In this paper we present an improved logic for analysing authentication properties of cryptographic protocols, based on the SVO logic of Syverson and van Oorschot. Such logics are useful in electronic commerce, among other areas. We have constructed this logic in order to simplify automation, and we describe an implementation using the Isabelle theorem-proving system, and a GUI tool based on this implementation. The tool is typically operated by opening a list of propositions intended to be true, and clicking one button. Since the rules form a clean framework, the logic is easily extensible. We also present in detail a proof of soundness, using Kripke possible-worlds semantics.

1. Introduction

During the past decade, there has been extensive interest in analysing the correctness of cryptographic protocols. Current interest in electronic commerce has added to this interest.

One set of solutions for analysing cryptographic protocols has been the BAN family of logics, beginning with the work of Burrows, Abadi, and Needham [1, 2] and continuing with the GNY logic of Gong, Needham, and Yahalom [3, 4]. These logics allow reasoning about authentication, but their complexity has resulted in problems with unsoundness and difficulty of implementation. In spite of the published soundness proof of BAN logic in [2], the BAN and GNY logics have been extensively criticised (e.g. [3, 4, 11]).

One of the more recent members of this family is the SVO logic of Syverson and van Oorschot [11]. The SVO logic unifies much of the previous work in an elegant framework, and is proved to be sound (which we view as essential, since we are particularly interested in the information security evaluation of cryptographic protocols at a high level). However, this logic is not suitable for automation with theorem provers such as Isabelle. Such automation is of great value for practical analysis of cryptographic protocols, and previous work has been done on automating the GNY logic (e.g. [5, 6, 7, 8, 9, 10]) which will discuss later in section 8.

In this paper we present an improvement to this logic which we call SVD logic. We have designed this logic with two things in mind: first, ease of implementation in the Isabelle theorem-proving system. This has required developing terminating proof tactics, and addressing all possible cases of circularity in the rules. We have demonstrated the success of our tactics by building a GUI tool which automates the proof process. Since the rules have been designed with automation in mind, the proof tactics required are fairly simple, and automation in a system other than Isabelle should be relatively easy. Because the set of rules forms a clean framework, it should also be easily extensible. Our tool is intended for users with no training in Isabelle, and is typically operated by opening a list of propositions intended to be true, and clicking one button.

Achieving an easily implemented set of rules has resulted in a relatively large number of rules, and so our second concern has been the soundness of the rules. We have achieved soundness by keeping the rules very simple (more complex rules have been derived as theorems) and giving the rules a more operational flavour, by casting them in terms of *operations* which a principal is capable of performing. We have developed a formal proof of soundness using Kripke possible-worlds semantics.

2. The Rules

We consider messages to be made up of *terms*, which are built out of atomic elements and the following operators:

$(x\,!\,y)$	Pairing
$\{x\}k$	Public- or symmetric-key encryption
$\{x\}^{\circ}k$	Symmetric-key decryption

There is an important reason why we consider terms rather than bit patterns: a theory based on bit patterns can result in undesirable behaviour. For example, an arbitrary bit pattern x could be viewed as $\{y\}k$ for some symmetric key k even when there is no reason to do so. There would then be no such thing as an atomic bit pattern, since this process could be continued ad infinitum. The fact that we can construct any arbitrary bit pattern also conflicts with the fact that any key is a bit pattern, but that we cannot construct every key. Using terms, we consider the term $\{y\}k$ to exist only when there was a term y which was actually encrypted with k by somebody.

We write $+k$ and $-k$ for the public and private keys associated with a public-key pair k. This gives us the following rules about encryption (where $\equiv$ denotes equality of terms):

K1 $\quad \{\{x\}k\}^{\circ}k \;\equiv\; x \qquad$ for symmetric keys k

K2 $\quad \{\{x\}{+k}\}{-k} \;\equiv\; x$

K3 $\quad \{\{x\}{-k}\}{+k} \;\equiv\; x$

Notice that the syntax $\{x\}k$ is ambiguous, but the type-checking facilities of Isabelle ensure that this does not cause any problems. The syntax $\hat{k}$ provides a useful shorthand for the inverse of a key which can be expressed easily using typed variables in Isabelle:

K4 $\quad \widehat{+k} \;\equiv\; -k$

K5 $\quad \widehat{-k} \;\equiv\; +k$

K6 $\quad \hat{k} \;\equiv\; k \qquad$ for symmetric keys k

One-way hash functions are treated as keys for which no inverse exists. We use the syntax $k\;\mathbf{DH}\;l$ for the Diffie-Hellman key generated from the public key pairs k and l. The syntax $\mathcal{P}\;\mathbf{PK}\;{+k}$ means that $+k$ is the public key for the principal $\mathcal{P}$, where a principal is any party (whether a person or a software agent) involved in a cryptographic protocol. The syntax $\mathcal{P} \overset{k}{\longleftrightarrow} \mathcal{Q}$ means that k is a good symmetric key for the principals $\mathcal{P}$ and $\mathcal{Q}$ (here $\Longleftrightarrow$ means 'if and only if'):

K7 $\quad k\;\mathbf{DH}\;l \;\equiv\; l\;\mathbf{DH}\;k$

K8 $\quad \mathcal{P} \overset{k}{\longleftrightarrow} \mathcal{Q} \Longleftrightarrow \mathcal{Q} \overset{k}{\longleftrightarrow} \mathcal{P}$

K9 $\quad \dfrac{\mathcal{P}\;\mathbf{PK}\;{+k}, \quad \mathcal{Q}\;\mathbf{PK}\;{+l}}{\mathcal{P} \overset{k\,\mathbf{DH}\,l}{\longleftrightarrow} \mathcal{Q}}$

The rule K9 says that a good symmetric key can be generated from two public key pairs (it corresponds to rule 5 in the SVO system, and K8 corresponds to rule 19). In this paper we use **PK** for public keys, whether used for authentication, encryption, or key agreement; but these can easily be distinguished by using different symbols if desired.

The rules T1–T4 define the obvious subterm relation on terms. We refer to this as the *strict* subterm relation, in contrast to the *loose* subterm relation, which will define later:

T1 $\quad \vdash x \;\subseteq\; x$

T2 $\quad \dfrac{z \;\subseteq\; x}{z \;\subseteq\; (x\,!\,y)}$

T3 $\quad \dfrac{z \;\subseteq\; y}{z \;\subseteq\; (x\,!\,y)}$

T4 $\quad \dfrac{y \;\subseteq\; x}{y \;\subseteq\; \{x\}k}$

The rules S1–S7 define what a principal **sees**. A principal sees what she receives in a message, what she has initially, what she can extract, and what she can construct. These rules correspond to rules 6–10 in the SVO system:

S1 $\quad \dfrac{\mathcal{P}\;\textbf{received}\;x}{\mathcal{P}\;\textbf{sees}\;x}$

S2 $\quad \dfrac{\mathcal{P}\;\textbf{initiallyhas}\;x}{\mathcal{P}\;\textbf{sees}\;x}$

S3 $\quad \dfrac{\mathcal{P}\;\textbf{sees}\;x, \quad x\;\rangle\mathcal{P}\rangle\;y}{\mathcal{P}\;\textbf{sees}\;y}$

S4 $\quad \dfrac{\mathcal{P}\;\textbf{sees}\;x, \quad \mathcal{P}\;\textbf{sees}\;y}{\mathcal{P}\;\textbf{sees}\;(x\,!\,y)}$

S5 $\quad \dfrac{\mathcal{P}\;\textbf{sees}\;x, \quad \mathcal{P}\;\textbf{sees}\;k}{\mathcal{P}\;\textbf{sees}\;\{x\}k}$

S6 $\quad \dfrac{\mathcal{P}\;\textbf{sees}\;x, \quad \mathcal{P}\;\textbf{sees}\;k}{\mathcal{P}\;\textbf{sees}\;\{x\}^{\circ}k}$

S7 $$\frac{\mathcal{P}\ \textbf{sees}\ +k, \quad \mathcal{P}\ \textbf{sees}\ -l}{\mathcal{P}\ \textbf{sees}\ k\ \textbf{DH}\ l}$$

The relation $\rangle\mathcal{P}\rangle$ indicates what the principal $\mathcal{P}$ can *extract*. This is essentially a special case of the subterm relation, where $\mathcal{P}$ has the necessary keys:

E1 $$\vdash x\ \rangle\mathcal{P}\rangle\ x$$

E2 $$\frac{x\ \rangle\mathcal{P}\rangle\ z}{(x\,!\,y)\ \rangle\mathcal{P}\rangle\ z}$$

E3 $$\frac{y\ \rangle\mathcal{P}\rangle\ z}{(x\,!\,y)\ \rangle\mathcal{P}\rangle\ z}$$

E4 $$\frac{x\ \rangle\mathcal{P}\rangle\ y, \quad \mathcal{P}\ \textbf{sees}\ \hat{k}}{\{x\}k\ \rangle\mathcal{P}\rangle\ y}$$

Formulating rules such as these in terms of operations (such as extraction) which a principal can perform makes it easier both to construct a set of easily automated rules and to prove the resultant rules sound.

To allow reasoning about **fresh** (recently created) terms, we have the following rules. These will be augmented by situation-specific axioms about the freshness of particular recently created atoms (i.e. nonces, fresh keys, etc.) for particular situations. These rules correspond to rules 16–17 in the SVO system:

F1 $$\frac{\textbf{fresh}\ x}{\textbf{fresh}\ (x\,!\,y)}$$

F2 $$\frac{\textbf{fresh}\ y}{\textbf{fresh}\ (x\,!\,y)}$$

F3 $$\frac{\textbf{fresh}\ x}{\textbf{fresh}\ \{x\}k}$$

F4 $$\frac{\textbf{fresh}\ x}{\textbf{fresh}\ \{x\}^{\circ}k}$$

The next five rules define when a principal can be said to have **said** something (we use the relation **says** when a principal has said something fresh, i.e. recently created). These rules correspond to rules 3, 4, 13, 14 and 18 in the SVO system. We assume that when good keys are used, we can believe that messages come from the principal that they appear to come from. That is to say, any third party that possesses the keys can be trusted not to use them. Also, when symmetric keys are used, a principal can distinguish between what it has sent and what its partner has sent.

This distinguishing between what a principal has sent and what its partner has sent (when symmetric keys are used) can be accomplished in a number of ways. For example, each principal can keep track of messages that they have sent (if a message is encrypted with a good symmetric key and we didn't send it, then the other party must have). It can also be accomplished by ensuring that messages at different steps of a protocol have different structures (if a message is encrypted with a good symmetric key and isn't of the right form to have come from us, then the other party must have sent it). Finally, it can also (but less desirably) be done by adding a one-bit indicator to a message (for a 0, it comes from the principal who initiated the protocol, and for a 1 it comes from the other party).

We indicate this distinguishing (as in SVO logic) with the notation $\{(y\,!\,\mathcal{Q})\}k$ for a message encrypted with k and distinguished as coming from $\mathcal{Q}$:

Z1 $$\frac{\mathcal{P}\ \textbf{said}\ x, \quad y \subseteq x}{\mathcal{P}\ \textbf{said}\ y}$$

Z2 $$\frac{\mathcal{P}\ \textbf{says}\ x, \quad y \subseteq x}{\mathcal{P}\ \textbf{says}\ y}$$

Z3 $$\frac{\mathcal{P}\ \textbf{said}\ x, \quad \textbf{fresh}\ x}{\mathcal{P}\ \textbf{says}\ x}$$

Z4 $$\frac{\mathcal{P}\ \textbf{received}\ x, \quad x\ \rangle\mathcal{P}\rangle\ \{(y\,!\,\mathcal{Q})\}k, \quad \mathcal{P} \overset{k}{\longleftrightarrow} \mathcal{Q}}{\mathcal{Q}\ \textbf{said}\ \{(y\,!\,\mathcal{Q})\}k}$$

Z5 $$\frac{\mathcal{P}\ \textbf{received}\ x, \quad x\ \rangle\mathcal{P}\rangle\ \{y\}{-l}, \quad \mathcal{Q}\ \textbf{PK}\ +l}{\mathcal{Q}\ \textbf{said}\ \{y\}{-l}}$$

The most important part of our system, as in SVO logic, is reasoning about *belief*. We wish principals to have *trust* in certain properties and conditions: this is in fact the main goal of cryptographic protocols. Trust is in fact a particular kind of belief, and is best analysed in terms of the logic of belief.

We use the following three general rules for the modal logic of belief (technically known as K4 or doxastic logic [13, 14, 15]), corresponding to rules 1 and 2 in the SVO system. Note that not all true statements are necessarily believed, not all beliefs are necessarily true, and we are using the $\models$ symbol to express belief rather than provability (for lack of a better symbol which can be used in both LaTeXand Isabelle). Note also that $\Longrightarrow$ means implication:

M1 $$\frac{\mathcal{P} \models a \Longrightarrow b, \quad \mathcal{P} \models a}{\mathcal{P} \models b}$$

M2 $$\frac{\mathcal{P} \models a}{\mathcal{P} \models \mathcal{P} \models a}$$

M3 $\quad \vdash \mathcal{P} \models a \qquad$ where a is a tautology

However, there are a number of areas where a principal believes things if and only if they are true, because of her personal knowledge. This includes objects that the principal has and operations that the principal can perform:

B1 $$\frac{\mathcal{P}\ \textbf{received}\ x}{\mathcal{P} \models \mathcal{P}\ \textbf{received}\ x}$$

B2 $$\frac{x\ \rangle\mathcal{P}\rangle\ y}{\mathcal{P} \models x\ \rangle\mathcal{P}\rangle\ y}$$

B3 $$\frac{\mathcal{P}\ \textbf{initiallyhas}\ x}{\mathcal{P} \models \mathcal{P}\ \textbf{initiallyhas}\ x}$$

B4 $$\frac{\mathcal{P}\ \textbf{PK}\ +k}{\mathcal{P} \models \mathcal{P}\ \textbf{PK}\ +k}$$

B5 $$\frac{\mathcal{P}\ \textbf{sees}\ x}{\mathcal{P} \models \mathcal{P}\ \textbf{sees}\ x}$$

We replace rules 11 and 12 of the SVO system by a somewhat more complicated set of rules. This is necessary because the attempt to give $\mathcal{P} \models a$ a meaning unrelated to a can lead to unsoundness. This can occur when there is conflict between one set of rules that define the truth of a and another set of rules that define the truth of $\mathcal{P} \models a$. If the semantics tries to enforce the falseness of $\mathcal{P} \models a$, but $\mathcal{P} \models a$ can be inferred from $\mathcal{P} \models b$ and $b \Longrightarrow a$, then the rules will be unsound.

We first provide a set of rules that say a principal **understands** something if it can be built up by construction or decryption from atomic elements:

U1 $\quad \mathcal{P}\ \textbf{understands}\ x \Longleftrightarrow \mathcal{P}\ \textbf{sees}\ x$ for atomic x

U2 $$\frac{\mathcal{P}\ \textbf{understands}\ x, \quad \mathcal{P}\ \textbf{understands}\ y}{\mathcal{P}\ \textbf{understands}\ (x\,!\,y)}$$

U3 $$\frac{\mathcal{P}\ \textbf{understands}\ x, \quad \mathcal{P}\ \textbf{sees}\ k}{\mathcal{P}\ \textbf{understands}\ \{x\}k}$$

U4 $$\frac{\mathcal{P}\ \textbf{understands}\ x, \quad \mathcal{P}\ \textbf{sees}\ k}{\mathcal{P}\ \textbf{understands}\ \{x\}^{\circ}k}$$

U5 $$\frac{\mathcal{P}\ \textbf{understands}\ x, \quad \mathcal{P}\ \textbf{sees}\ \{x\}k, \quad \mathcal{P}\ \textbf{sees}\ \hat{k}}{\mathcal{P}\ \textbf{understands}\ \{x\}k}$$

B6 $$\frac{\mathcal{P}\ \textbf{understands}\ x}{\mathcal{P} \models \mathcal{P}\ \textbf{understands}\ x}$$

The rules U3 and U5 produce similar conclusions, but for different reasons. With rule U3, $\mathcal{P}$ understands $\{x\}k$ because she can construct it from things she understands, while with rule U5 she understands $\{x\}k$ because she can decrypt it to give something which she understands. In the special case of symmetric keys, where $\hat{k} \equiv k$, U5 becomes a special case of U3.

We also have a relation $\succ\!\mathcal{P}\!\succ$ which indicates that a principal recognises something as being a subterm of another (another example of a relation expressed in terms of an operation that a principal can perform). This relation is stronger than the relation $\supseteq$ but weaker than $\rangle\mathcal{P}\rangle$. The difference lies in the rule E9, which says that a principal can recognise subterms of $\{x\}k$ if she understands $\{x\}k$ through construction (as in rule U3). For example, if h is a one-way hash function (i.e. a key without an inverse) which $\mathcal{P}$ has, and $\mathcal{P}$ understands x, then $\{x\}h\ \succ\!\mathcal{P}\!\succ\ x$ (we will use this in our first example, later in the paper):

E5 $\quad \vdash x\ \succ\!\mathcal{P}\!\succ\ x$

E6 $$\frac{x\ \succ\!\mathcal{P}\!\succ\ z}{(x\,!\,y)\ \succ\!\mathcal{P}\!\succ\ z}$$

E7 $$\frac{y\ \succ\!\mathcal{P}\!\succ\ z}{(x\,!\,y)\ \succ\!\mathcal{P}\!\succ\ z}$$

E8 $$\frac{x\ \succ\!\mathcal{P}\!\succ\ y, \quad \mathcal{P}\ \textbf{sees}\ \hat{k}}{\{x\}k\ \succ\!\mathcal{P}\!\succ\ y}$$

E9 $$\frac{x\ \succ\!\mathcal{P}\!\succ\ y, \quad \mathcal{P}\ \textbf{sees}\ k, \quad \mathcal{P}\ \textbf{understands}\ x}{\{x\}k\ \succ\!\mathcal{P}\!\succ\ y}$$

B7 $$\frac{x\ \succ\!\mathcal{P}\!\succ\ y}{\mathcal{P} \models x\ \succ\!\mathcal{P}\!\succ\ y}$$

We use this to provide two variations of rules Z1 and Z2 that use the relation $\succ\!\mathcal{P}\!\succ$ to indicate that a principal recognises a subterm relationship. This is necessary because, to ensure soundness, we have given no interpretation to $\mathcal{Q} \models y \subseteq x$:

Z6 $$\frac{\mathcal{Q} \models \mathcal{P}\ \textbf{said}\ x, \quad x\ \succ\!\mathcal{Q}\!\succ\ y}{\mathcal{Q} \models \mathcal{P}\ \textbf{said}\ y}$$

Z7 $$\frac{\mathcal{Q} \models \mathcal{P}\ \textbf{says}\ x, \quad x\ \succ\!\mathcal{Q}\!\succ\ y}{\mathcal{Q} \models \mathcal{P}\ \textbf{says}\ y}$$

Finally, we introduce terms of the form **assert** x which assert a proposition, and say that a principal believes a proposition if she believes it comes from a trustworthy source (these assertions are usually implicit, based on the context of a message in a particular protocol). This corresponds to the more limited rule 15 in the SVO system, which is only valid if the trusted source does not make an error. We express our version of the rule in terms of belief to explicitly cater for the case where a source is trusted (in the sense that some principals believe it) but can still make false statements:

$$\text{C1} \quad \frac{\mathcal{Q} \models \mathcal{P}\ \textbf{controls}\ x \qquad \mathcal{Q} \models \mathcal{P}\ \textbf{says}\ (\textbf{assert}\ x)}{\mathcal{Q} \models x}$$

$$\text{C2} \quad \frac{\mathcal{Q} \models \mathcal{P}\ \textbf{controlled}\ x \qquad \mathcal{Q} \models \mathcal{P}\ \textbf{said}\ (\textbf{assert}\ x)}{\mathcal{Q} \models x}$$

There are no rules to infer $\mathcal{P}$ **controls** x (i.e. $\mathcal{P}$ can be trusted when she has recently said x) or $\mathcal{P}$ **controlled** x ($\mathcal{P}$ can be trusted when she has ever said x) since these two statements occur only in situation-specific axioms.

Note that if $\mathcal{Q}$ trusts a principal who makes false statements, then she will have false beliefs as a result.

3. The Implementation

We have implemented this system in Isabelle firstly by introducing a logic and a set of tactics based on the sequent calculus and on rules derived from the belief rules M1–M3. Secondly we have attempted to produce a terminating proof tactic, by eliminating circularity from the rules. There are two problems here: on the one hand, rules C1 and C2, which are totally unsuitable for automatic backwards proof (because the hypotheses are instances of the conclusion). In the GUI tool we have developed, these rules are applied manually, by clicking a button. They can also be applied automatically (using forwards proof) if the second hypothesis is explicitly listed by the user as an intermediate goal. The other cause of circularity is rules E1–E4 and S1–S7 which depend on each other. We have addressed this by replacing E1–E4 by an equivalent set of rules:

$$\text{X1} \quad x \rangle \mathcal{P} \rangle\ y \Longleftrightarrow x \rangle \mathcal{P}, [\,] \rangle\ y$$

$$\text{X2} \quad \vdash x \rangle \mathcal{P}, [\,] \rangle\ x$$

$$\text{X3} \quad \frac{\mathcal{P}\ \textbf{sees}\ k, \quad x \rangle \mathcal{P}, \lambda \rangle\ x}{x \rangle \mathcal{P}, [k|\lambda] \rangle\ x}$$

$$\text{X4} \quad \frac{x \rangle \mathcal{P}, \lambda \rangle\ z}{(x \,!\, y) \rangle \mathcal{P}, \lambda \rangle\ z}$$

$$\text{X5} \quad \frac{y \rangle \mathcal{P}, \lambda \rangle\ z}{(x \,!\, y) \rangle \mathcal{P}, \lambda \rangle\ z}$$

$$\text{X6} \quad \frac{x \rangle \mathcal{P}, [\hat{k}|\lambda] \rangle\ y}{\{x\}k \rangle \mathcal{P}, \lambda \rangle\ y}$$

Here $[k|\lambda]$ represents a list of keys with head k and tail λ, and $[\,]$ represents an empty list. The notation $x \rangle \mathcal{P}, [k|\lambda] \rangle\ y$ means that the principal $\mathcal{P}$ can extract y from x, and that λ is a list of keys which are needed to do this. These rules have precisely the same effect as the rules E1–E4, but assist automated reasoning by delaying the search for $\mathcal{P}$ **sees** $\hat{k}$ until we know that we need it. Non-termination can still occur, but *only* in pathological cases where a key occurs in places where it is itself necessary for extraction, for example $\{\hat{k}\}k$ or $(\{\hat{k_1}\}k_2 \,!\, \{\hat{k_2}\}k_1)$.

More formally, define $k_1 \leq^{\circ}_{\mathcal{P}} k_2$ if and only if in a term e received or initially had by $\mathcal{P}$, there is a subterm $\{e'\}k_2$ where $\hat{k_1}$ occurs in e' (i.e. $\hat{k_1} \subseteq e'$ and $\{e'\}k_2 \subseteq e$), and define $\leq_{\mathcal{P}}$ as the transitive (but not reflexive) closure of $\leq^{\circ}_{\mathcal{P}}$. The relation $k_1 \leq_{\mathcal{P}} k_2$ formalises the concept that k_2 is needed to extract $\hat{k_1}$, and hence $k \leq_{\mathcal{P}} k$ formalises the concept that $\hat{k}$ occurs in places where it is itself necessary for extraction.

Now consider the case where, for example, we have $\mathcal{P}$ **received** e, and we are trying to prove $\mathcal{P}$ **sees** $\hat{k}$ by backwards proof using S1 and S3 (which requires $e \rangle \mathcal{P} \rangle\ \hat{k}$). If it is the case that $k \leq_{\mathcal{P}} k$, backwards proof will not terminate, since application of rules X1–X6 produces $\mathcal{P}$ **sees** $\hat{k}$ as a subgoal. Conversely, if it is *not* the case that $k \leq_{\mathcal{P}} k$, then application of rules X4–X6 results in one of two possible cases:

1. (i) $\{e'\}k$ is not a subterm of e, and rule X6 will not add $\hat{k}$ to the list of keys λ, or

2. (ii) $\{e'\}k$ is a subterm of e for some e', but $\hat{k}$ (or a key required to find $\hat{k}$) does not occur in e' and hence rule X3 is never applied for $\hat{k}$.

In both cases, non-termination will not result. In other words, non-termination occurs if and only if $k \leq_{\mathcal{P}} k$ for some k.

In addition to the rules X1–X6, we also define a *loose* subterm relation, which differs from the *strict* subterm relation ($\subseteq$) in the last three rules:

$$\text{T5} \quad \vdash x \sqsubseteq x$$

$$\text{T6} \quad \frac{z \sqsubseteq x}{z \sqsubseteq (x \,!\, y)}$$

T7 $$\frac{z \sqsubseteq y}{z \sqsubseteq (x\,!\,y)}$$

T8 $$\frac{y \sqsubseteq x}{y \sqsubseteq \{x\}k}$$

T9 $$\frac{y \sqsubseteq k}{y \sqsubseteq \{x\}k}$$

T10 $\vdash k \sqsubseteq +k$

T11 $\vdash k \sqsubseteq -k$

If $+k$ and $-k$ are a public/private key pair, k can be considered to be the information from which the pair is derived, or else $k \sqsubseteq +k$ can be interpreted purely syntactically.

The relation $\sqsubseteq$ is used to define a new quantifier $\exists x{\sqsubseteq}y.\ a$, equivalent to $\exists x.\ x \sqsubseteq y \ \wedge\ a$. Within the sequent calculus, we can write a terminating and complete tactic for this quantifier (by the obvious induction on the structure of y), which we cannot do for the existential quantifier in general.

Using this new quantifier, we can replace rules Z1–Z7 by eight new derived rules which are more suitable for automatic backwards proof:

D1 $$\frac{\begin{array}{l}\mathcal{P}\ \textbf{received}\ x\\ \exists y{\sqsubseteq}x.\ \exists k{\sqsubseteq}x.\ \mathcal{P} \xleftrightarrow{k} \mathcal{Q}\\ \qquad\qquad \wedge\ x\ \rangle\mathcal{P}\rangle\ \{(y\,!\,\mathcal{Q})\}k\ \wedge\ \textbf{fresh}\ y\\ \qquad\qquad \wedge\ z\ \subseteq\ \{(y\,!\,\mathcal{Q})\}k\end{array}}{\mathcal{Q}\ \textbf{says}\ z}$$

D2 $$\frac{\begin{array}{l}\mathcal{P}\ \textbf{received}\ x\\ \exists y{\sqsubseteq}x.\ \exists l{\sqsubseteq}x.\ \mathcal{Q}\ \textbf{PK}\ {+l}\ \wedge\ x\ \rangle\mathcal{P}\rangle\ \{y\}{-l}\\ \qquad\qquad \wedge\ \textbf{fresh}\ y\ \wedge\ z\ \subseteq\ \{y\}{-l}\end{array}}{\mathcal{Q}\ \textbf{says}\ z}$$

D3 $$\frac{\begin{array}{l}\mathcal{P}\ \textbf{received}\ x\\ \exists y{\sqsubseteq}x.\ \exists k{\sqsubseteq}x.\ \mathcal{P} \xleftrightarrow{k} \mathcal{Q}\\ \qquad\qquad \wedge\ x\ \rangle\mathcal{P}\rangle\ \{(y\,!\,\mathcal{Q})\}k\\ \qquad\qquad \wedge\ z\ \subseteq\ \{(y\,!\,\mathcal{Q})\}k\end{array}}{\mathcal{Q}\ \textbf{said}\ z}$$

D4 $$\frac{\begin{array}{l}\mathcal{P}\ \textbf{received}\ x\\ \exists y{\sqsubseteq}x.\ \exists l{\sqsubseteq}x.\ \mathcal{Q}\ \textbf{PK}\ {+l}\ \wedge\ x\ \rangle\mathcal{P}\rangle\ \{y\}{-l}\\ \qquad\qquad \wedge\ z\ \subseteq\ \{y\}{-l}\end{array}}{\mathcal{Q}\ \textbf{said}\ z}$$

D5 $$\frac{\begin{array}{l}\mathcal{P}\ \textbf{received}\ x\\ \exists y{\sqsubseteq}x.\ \exists k{\sqsubseteq}x.\ \mathcal{P} \models \mathcal{P} \xleftrightarrow{k} \mathcal{Q}\\ \qquad\qquad \wedge\ x\ \rangle\mathcal{P}\rangle\ \{(y\,!\,\mathcal{Q})\}k\\ \qquad\qquad \wedge\ \mathcal{P} \models \textbf{fresh}\ y\\ \qquad\qquad \wedge\ \{(y\,!\,\mathcal{Q})\}k\ \succ\mathcal{P}\succ\ z\end{array}}{\mathcal{P} \models \mathcal{Q}\ \textbf{says}\ z}$$

D6 $$\frac{\begin{array}{l}\mathcal{P}\ \textbf{received}\ x\\ \exists y{\sqsubseteq}x.\ \exists l{\sqsubseteq}x.\ \mathcal{P} \models \mathcal{Q}\ \textbf{PK}\ {+l}\\ \qquad\qquad \wedge\ x\ \rangle\mathcal{P}\rangle\ \{y\}{-l}\\ \qquad\qquad \wedge\ \mathcal{P} \models \textbf{fresh}\ y\\ \qquad\qquad \wedge\ \{y\}{-l}\ \succ\mathcal{P}\succ\ z\end{array}}{\mathcal{P} \models \mathcal{Q}\ \textbf{says}\ z}$$

D7 $$\frac{\begin{array}{l}\mathcal{P}\ \textbf{received}\ x\\ \exists y{\sqsubseteq}x.\ \exists k{\sqsubseteq}x.\ \mathcal{P} \models \mathcal{P} \xleftrightarrow{k} \mathcal{Q}\\ \qquad\qquad \wedge\ x\ \rangle\mathcal{P}\rangle\ \{(y\,!\,\mathcal{Q})\}k\\ \qquad\qquad \wedge\ \{(y\,!\,\mathcal{Q})\}k\ \succ\mathcal{P}\succ\ z\end{array}}{\mathcal{P} \models \mathcal{Q}\ \textbf{said}\ z}$$

D8 $$\frac{\begin{array}{l}\mathcal{P}\ \textbf{received}\ x\\ \exists y{\sqsubseteq}x.\ \exists l{\sqsubseteq}x.\ \mathcal{P} \models \mathcal{Q}\ \textbf{PK}\ {+l}\\ \qquad\qquad \wedge\ x\ \rangle\mathcal{P}\rangle\ \{y\}{-l}\\ \qquad\qquad \wedge\ \{y\}{-l}\ \succ\mathcal{P}\succ\ z\end{array}}{\mathcal{P} \models \mathcal{Q}\ \textbf{said}\ z}$$

The soundness of these rules was demonstrated by deriving them from Z1–Z7 in Isabelle. Their completeness (with respect to Z1–Z7) can be demonstrated by a simple induction on proof trees built from Z1–Z7. In particular, it is clear that the relations $y \sqsubseteq x$ etc. must hold if e.g. $x\ \rangle\mathcal{P}\rangle\ \{(y\,!\,\mathcal{Q})\}k$ so that the use of the bounded quantifier does not lose generality.

The sequents we use have the form $a_1, \ldots, a_n \vdash b$ where $a_1, \ldots, a_n$ are situation-specific axioms, and b is what we want to prove. The sequent calculus proof tactics are simplified by the fact that the statements $\mathcal{P}$ **initiallyhas** x or $\mathcal{P}$ **received** x occur only in situation-specific axioms: they are not created by the rules. Thus they need only be searched for in the left-hand side of a sequent. The sequent calculus version of rule D1 thus has the form (here α, β, γ, δ represent sets of statements):

D1′ $$\frac{\alpha,\ \mathcal{P}\ \textbf{received}\ x,\ \beta \vdash \gamma,\ \exists y{\sqsubseteq}x.\ \ldots,\ \delta}{\alpha,\ \mathcal{P}\ \textbf{received}\ x,\ \beta \vdash \gamma,\ \mathcal{Q}\ \textbf{says}\ z,\ \delta}$$

The cases of D2–D8 are similar.

4. First Example

In this paper we consider two examples of the use of SVD logic. Our first example involves the encrypted and authenticated transfer of a message M between two parties $\mathcal{A}$ and $\mathcal{B}$, with the participation of a trusted third party $\mathcal{C}$. Such a transfer occurs in e.g. electronic commerce. We begin with a series of axioms for this specific situation (we use some fairly obvious shorthand conventions, and h and k are a hash key and a new symmetric session key respectively). We also use $*$ as a wild-card pattern (our GUI tool understands such patterns):

$$\mathcal{A}\ \mathbf{PK}\ {+a},\ \mathcal{B}\ \mathbf{PK}\ {+b},\ \mathcal{C}\ \mathbf{PK}\ {+c}$$

$$\mathcal{A}\ \mathbf{initiallyhas}\ h, k, +a, -a, +c, \mathcal{A}, \mathcal{B}, \mathcal{C}$$

$$\mathcal{B}\ \mathbf{initiallyhas}\ h, +b, -b, +c, \mathcal{A}, \mathcal{B}, \mathcal{C}$$

$$\mathcal{A} \models \mathcal{C}\ \mathbf{PK}\ {+c}$$

$$\mathcal{B} \models \mathcal{C}\ \mathbf{PK}\ {+c}$$

$$\mathcal{A} \models \mathcal{C}\ \mathbf{controlled} * \mathbf{PK} *$$

$$\mathcal{B} \models \mathcal{C}\ \mathbf{controlled} * \mathbf{PK} *$$

The protocol involves two certificates from $\mathcal{C}$, which assert the existence and goodness of public keys for $\mathcal{A}$ and $\mathcal{B}$ (authenticated by the private key for $\mathcal{C}$):

$$\mathcal{A}\ \mathbf{received}\ \theta = \{(+a\,!\,(\mathbf{assert}\ \mathcal{A}\ \mathbf{PK}\ {+a}))\}{-c}$$

$$\mathcal{A}\ \mathbf{received}\ \{(+b\,!\,(\mathbf{assert}\ \mathcal{B}\ \mathbf{PK}\ {+b}))\}{-c}$$

We use θ as a shorthand for the first certificate, which $\mathcal{A}$ will pass on to $\mathcal{B}$. Finally there is a complex message from $\mathcal{A}$ to $\mathcal{B}$, which contains the message M, a digital signature of M, and the certificate θ, all encrypted with the session key k. The session key is attached, encrypted with the public key for $\mathcal{B}$:

$$\mathcal{B}\ \mathbf{received}\ (\{(M\,!\,\{\{M\}h\}{-a}\,!\,\theta)\}k\,!\,\{k\}{+b})$$

We are then able to prove the following statements (in particular that $\mathcal{B}$ believes that the message M indeed came from $\mathcal{A}$):

$\mathcal{A} \models \mathcal{C}\ \mathbf{said}\ (\mathbf{assert}\ \mathcal{B}\ \mathbf{PK}\ {+b})$	by D8, E5–E8
$\mathcal{A} \models \mathcal{B}\ \mathbf{PK}\ {+b}$	by C2
$\mathcal{A}\ \mathbf{sees}\ {+b}$	by E1–E4, S1, S3
$\mathcal{B}\ \mathbf{sees}\ k$	by E1–E4, S1, S3
$\mathcal{B}\ \mathbf{sees}\ M$	by E1–E4, S1, S3
$\mathcal{B}\ \mathbf{sees}\ {+a}$	by E1–E4, S1, S3
$\mathcal{B} \models \mathcal{C}\ \mathbf{said}\ (\mathbf{assert}\ \mathcal{A}\ \mathbf{PK}\ {+a})$	by D8, E1–E8
$\mathcal{B} \models \mathcal{A}\ \mathbf{PK}\ {+a}$	by C2
$\mathcal{B} \models \mathcal{A}\ \mathbf{said}\ M$	by D8, E1–E9

The last step uses the E8 and E9 rules, and the fact that $\mathcal{B}$ recognises M inside the digital signature, i.e. $\{\{M\}h\}{-a}\ \succ\!\mathcal{B}\!\succ\ M$. For the purposes of this analysis, the message M is considered to be an atom, since it has no constituent parts as far as our logic is concerned.

5. Second Example

Our second example involves the first five steps of the Needham-Schroeder key exchange protocol, as modified to exclude the original design error. Two parties $\mathcal{A}$ and $\mathcal{B}$ wish to communicate privately with each other, using a key generated by a trusted third party $\mathcal{C}$. The axioms for this specific situation are as follows ($k_{\mathcal{AC}}$, $k_{\mathcal{BC}}$, and $k_{\mathcal{AB}}$ are symmetric keys, and $N_{\mathcal{A}}$ and $N_{\mathcal{B}}$ are nonces, i.e. fresh atomic terms):

$$\mathcal{A} \overset{k_{\mathcal{AC}}}{\longleftrightarrow} \mathcal{C},\ \mathcal{B} \overset{k_{\mathcal{BC}}}{\longleftrightarrow} \mathcal{C},\ \mathcal{A} \overset{k_{\mathcal{AB}}}{\longleftrightarrow} \mathcal{B}$$

$$\mathcal{A}\ \mathbf{initiallyhas}\ k_{\mathcal{AC}}, N_{\mathcal{A}}, \mathcal{A}, \mathcal{B}, \mathcal{C}$$

$$\mathcal{B}\ \mathbf{initiallyhas}\ k_{\mathcal{BC}}, N_{\mathcal{B}}, \mathcal{A}, \mathcal{B}, \mathcal{C}$$

$$\mathbf{fresh}\ N_{\mathcal{A}}, N_{\mathcal{B}}$$

$$\mathcal{A} \models \mathcal{A} \overset{k_{\mathcal{AC}}}{\longleftrightarrow} \mathcal{C}$$

$$\mathcal{B} \models \mathcal{B} \overset{k_{\mathcal{BC}}}{\longleftrightarrow} \mathcal{C}$$

$$\mathcal{A} \models \mathbf{fresh}\ N_{\mathcal{A}}$$

$$\mathcal{B} \models \mathbf{fresh}\ N_{\mathcal{B}}$$

$$\mathcal{A} \models \mathcal{C}\ \mathbf{controls} * \overset{*}{\longleftrightarrow} *$$

$$\mathcal{B} \models \mathcal{C}\ \mathbf{controls} * \overset{*}{\longleftrightarrow} *$$

The principal $\mathcal{A}$ initiates the protocol with a message to $\mathcal{B}$. She responds with a message containing a nonce $N_{\mathcal{B}}$, which is then passed on to the trusted third party, along with a nonce for $\mathcal{A}$:

$$\mathcal{B}\ \mathbf{received}\ \mathcal{A}$$

$$\mathcal{A}\ \mathbf{received}\ \{(\mathcal{A}\,!\,N_{\mathcal{B}})\}k_{\mathcal{BC}}$$

$$\mathcal{C}\ \mathbf{received}\ (\mathcal{A}\,!\,\mathcal{B}\,!\,N_{\mathcal{A}}\,!\,\{(\mathcal{A}\,!\,N_{\mathcal{B}})\}k_{\mathcal{BC}})$$

This is followed by a complex message from the trusted third party to $\mathcal{A}$, part of which is in turn passed on to $\mathcal{B}$. This message includes the key $k_{\mathcal{AB}}$, an assertion as to its goodness, and the appropriate nonces to guarantee freshness:

$$\theta = \{(N_{\mathcal{B}}\,!\,k_{\mathcal{AB}}\,!\,\mathbf{assert}\ \mathcal{A} \overset{k_{\mathcal{AB}}}{\longleftrightarrow} \mathcal{B}\,!\,\mathcal{C})\}k_{\mathcal{BC}}$$

$\mathcal{A}$ **received** $\{(N_{\mathcal{A}} \,!\, k_{\mathcal{AB}} \,!\, \textbf{assert}\ \mathcal{A} \overset{k_{\mathcal{AB}}}{\longleftrightarrow} \mathcal{B} \,!\, \theta \,!\, \mathcal{C})\}k_{\mathcal{AC}}$

$\mathcal{B}$ **received** θ

We are then able to prove that both parties have the key $k_{\mathcal{AB}}$ (and believe that it is good):

$\mathcal{A}$ **sees** $k_{\mathcal{AB}}$	by E1–E4, S1, S3
$\mathcal{A} \models \mathcal{C}$ **says** (**assert** $\mathcal{A} \overset{k_{\mathcal{AB}}}{\longleftrightarrow} \mathcal{B}$)	by D5, E1–8, F1–4
$\mathcal{A} \models \mathcal{A} \overset{k_{\mathcal{AB}}}{\longleftrightarrow} \mathcal{B}$	by C1
$\mathcal{B}$ **sees** $k_{\mathcal{AB}}$	by E1–E4, S1, S3
$\mathcal{B} \models \mathcal{C}$ **says** (**assert** $\mathcal{A} \overset{k_{\mathcal{AB}}}{\longleftrightarrow} \mathcal{B}$)	by D5, E1–8, F1–4
$\mathcal{B} \models \mathcal{A} \overset{k_{\mathcal{AB}}}{\longleftrightarrow} \mathcal{B}$	by C1

6. The GUI Tool

We have developed a GUI tool for this logic which we call C3PO. A screen dump of the tool is included in the appendix. The tool has been developed in the **Tcl** language using the **TK** graphical toolkit. The tool maintains a list of propositions $a_1, \ldots, a_n$ which include the situation-specific axioms and the statements that have been proved so far. When the user enters a statement b to be proved, the sequent $a_1, \ldots, a_n \vdash b$ is passed to Isabelle for proving. The **Expect** interfacing extensions to **Tcl/TK** were used to handle communication with Isabelle.

The tool also allows automatic processing of a to-do list of statements, producing a log of successes and failures. Pattern-matching within the tool is used to implement the C1 and C2 rules, and these are applied automatically when a suitable sub-goal is proved. For example, the lists of statements in the two examples above can be used as to-do lists, and can be successfully processed by the tool without any manual intervention by the user. That is, once the to-do list is opened, the only thing the user needs to do is click the **Start Proof** button. If desired, the user can also enter additional statements and attempt to prove them. The time taken to process most examples is a few minutes.

7. Semantics

We prove the soundness of our system using Kripke (possible worlds) semantics [12, 13, 14, 15]. Each (possible) world contains 12 components. Some components are indexed by principals ($\mathcal{P}$) or time ($t \geq 0$ an integer, since we consider time to increase in fixed steps every time a significant event occurs). The rules K1–K7 and T1–T11 continue to apply to terms. The components are:

1. Sets N_t of atoms (nonces) created at time t.
2. Sets F_t of terms which are fresh at time t. We assume times up to a fixed limit Δ are fresh. These sets are defined by (i.e. are minimal sets satisfying):
$$N_t, N_{t-1}, \ldots, N_{t-\Delta} \subseteq F_t$$
$$x \in F_t \Longrightarrow (x \,!\, y) \in F_t, (y \,!\, x) \in F_t$$
$$x \in F_t \Longrightarrow \{x\}k \in F_t, \{x\}^{\circ}k \in F_t$$
The statement **fresh** x is true in world w at time t (we use the notation $\langle w, t\rangle : \textbf{fresh}\ x$) if and only if $x \in F_t$, where the world w is a 12-tuple of the form $(\{N_t\}, \{F_t\}, M, \{S_{\mathcal{P}t}\}, \{\hookrightarrow_{\mathcal{P}t}\}, \{G_t\}, \{H_t\}, \{Z_{\mathcal{P}t}\}, \{U_{\mathcal{P}t}\}, \{\mapsto_{\mathcal{P}t}\}, \{C_{\mathcal{P}t}\}, \{D_{\mathcal{P}t}\})$. It is clear that the freshness rules F1–F4 are sound on this interpretation.
3. A message history M containing elements of the form $(\mathcal{P} \rightharpoonup_t \mathcal{Q}\colon x), \mathcal{P} \neq \mathcal{Q}$ ($\mathcal{P}$ sends x to $\mathcal{Q}$ at time t). We define $\langle w, t\rangle : \mathcal{Q}$ **received** x if and only if $(\mathcal{P} \rightharpoonup_{t'} \mathcal{Q}\colon x) \in M$ for some $t' < t$ (next-step receipt).
4. Sets $S_{\mathcal{P}t}$ of terms seen by $\mathcal{P}$ at time t. These sets are defined by the choice of $S_{\mathcal{P}0}$ and:
$$S_{\mathcal{P}t'} \subseteq S_{\mathcal{P}t} \qquad \text{for } t' < t$$
$$x \in S_{\mathcal{P}t}, y \in S_{\mathcal{P}t} \Longrightarrow (x \,!\, y) \in S_{\mathcal{P}t}$$
$$x \in S_{\mathcal{P}t}, k \in S_{\mathcal{P}t} \Longrightarrow \{x\}k \in S_{\mathcal{P}t}, \{x\}^{\circ}k \in S_{\mathcal{P}t}$$
$$+k \in S_{\mathcal{P}t}, -l \in S_{\mathcal{P}t} \Longrightarrow k\ \textbf{DH}\ l \in S_{\mathcal{P}t}$$
$$x \in S_{\mathcal{P}t}, x \hookrightarrow_{\mathcal{P}t} y \Longrightarrow y \in S_{\mathcal{P}t}$$
$$(\mathcal{P} \rightharpoonup_{t'} \mathcal{Q}\colon x) \in M, t' < t \Longrightarrow x \in S_{\mathcal{P}t'}, x \in S_{\mathcal{Q}t}$$
5. Reflexive and transitive binary relations $\hookrightarrow_{\mathcal{P}t}$ describing what can be extracted at time t, defined by:
$$(x \,!\, y) \hookrightarrow_{\mathcal{P}t} x$$
$$(x \,!\, y) \hookrightarrow_{\mathcal{P}t} y$$
$$\{x\}k \hookrightarrow_{\mathcal{P}t} x \qquad \text{if } \hat{k} \in S_{\mathcal{P}t}$$
We define $\langle w, t\rangle : \mathcal{P}$ **initiallyhas** x if and only if $x \in S_{\mathcal{P}0}$ i.e. in the initial world. We also define $\langle w, t\rangle : \mathcal{P}$ **sees** x if and only if $x \in S_{\mathcal{P}t}$, and we define $\langle w, t\rangle : x \;\rangle\mathcal{P}\rangle\; y$ if and only if $x \hookrightarrow_{\mathcal{P}t} y$. It is clear that the rules E1–E4 and S1–S7 are sound on this interpretation.

6. Sets G_t of triples $(\mathcal{P}, k, \mathcal{Q})$ of symmetric keys which are good at time t. The sets must satisfy our assumptions about how good keys are used:

   (a) $(\mathcal{P}, k, \mathcal{Q}) \in G_t$ if and only if $(\mathcal{Q}, k, \mathcal{P}) \in G_t$.

   (b) If $(\mathcal{P}, k, \mathcal{Q}) \in G_t$ then every term of the form $\{(x\,!\,\mathcal{P})\}k$ first occurs in the message history M as $(\mathcal{P} \rightharpoonup_{t'} \mathcal{R}\colon e)$ for some $\mathcal{R}$ and $e \supseteq \{(x\,!\,\mathcal{P})\}k$.

   We define $\langle w, t\rangle : \mathcal{P} \overset{k}{\longleftrightarrow} \mathcal{Q}$ if and only if $(\mathcal{P}, k, \mathcal{Q}) \in G_t$. It is clear that the rule K8 is sound on this interpretation.

7. Sets H_t of pairs $(\mathcal{P}, k)$ of public keys which are good at time t. These sets must satisfy our assumptions about how good keys are used:

   (a) If $(\mathcal{P}, k) \in H_t$ then every term of the form $\{x\}{-k}$ first occurs in the message history M as $(\mathcal{P} \rightharpoonup_{t'} \mathcal{R}\colon e)$ for some $\mathcal{R}$ and $e \supseteq \{x\}{-k}$.

   (b) If $(\mathcal{P}, k) \in H_t$ and $(\mathcal{Q}, l) \in H_t$ then $(\mathcal{P}, k\ \mathbf{DH}\ l, \mathcal{Q}) \in G_t$

   We define $\langle w, t\rangle : \mathcal{P}\ \mathbf{PK}\ {+k}$ if and only if $(\mathcal{P}, k) \in H_t$. It is clear that the rule K9 is sound on this interpretation.

8. Sets $Z_{\mathcal{P}t}$ of terms said by $\mathcal{P}$ at time t, such that $x \in Z_{\mathcal{P}t}$ if and only if there is some term $e \supseteq x$ such that e first occurs in the message history M as $(\mathcal{P} \rightharpoonup_{t'} \mathcal{Q}\colon e')$ for some $\mathcal{Q}$, $e' \supseteq e$, and $t' < t$. We define $\langle w, t\rangle : \mathcal{P}$ **said** x if and only if $x \in Z_{\mathcal{P}t}$, and $\langle w, t\rangle : \mathcal{P}$ **says** x if only if $x \in Z_{\mathcal{P}t} \cap F_t$. Soundness of the rules Z1 and Z2 follow from the $e \supseteq x$ condition. Soundness of the rule Z3 follows from the reference to F_t. Soundness of the rules Z4 and Z5 follow from the definitions of G_t and H_t.

9. Sets $U_{\mathcal{P}t}$ of terms understood by $\mathcal{P}$ at time t. These sets are defined by:

$$U_{\mathcal{P}t'} \subseteq U_{\mathcal{P}t} \qquad \text{for } t' < t$$

$$x \in U_{\mathcal{P}t} \Longleftrightarrow x \in S_{\mathcal{P}t} \qquad \text{for atomic } x$$

$$x \in U_{\mathcal{P}t}, y \in U_{\mathcal{P}t} \Longrightarrow (x\,!\,y) \in U_{\mathcal{P}t}$$

$$x \in U_{\mathcal{P}t}, k \in S_{\mathcal{P}t} \Longrightarrow \{x\}k \in U_{\mathcal{P}t}, \{x\}^{\circ}k \in U_{\mathcal{P}t}$$

$$x \in U_{\mathcal{P}t}, \{x\}k \in S_{\mathcal{P}t}, \hat{k} \in S_{\mathcal{P}t} \Longrightarrow \{x\}k \in U_{\mathcal{P}t}$$

   We define $\langle w, t\rangle : \mathcal{P}$ **understands** x if and only if $x \in U_{\mathcal{P}t}$. It is clear that the rules U1–U5 are sound on this interpretation. It also follows from the definition of $S_{\mathcal{P}t}$ that $U_{\mathcal{P}t} \subseteq S_{\mathcal{P}t}$.

10. Reflexive and transitive binary relations $\mapsto_{\mathcal{P}t}$ describing what can be recognised as a subterm at time t, defined by:

$$(x\,!\,y) \mapsto_{\mathcal{P}t} x$$

$$(x\,!\,y) \mapsto_{\mathcal{P}t} y$$

$$\{x\}k \mapsto_{\mathcal{P}t} x \qquad \text{if } \hat{k} \in S_{\mathcal{P}t}$$

$$\{x\}k \mapsto_{\mathcal{P}t} x \qquad \text{if } k \in S_{\mathcal{P}t} \text{ and } x \in U_{\mathcal{P}t}$$

   We define $\langle w, t\rangle : x \succ\!\mathcal{P}\!\succ y$ if and only if $x \mapsto_{\mathcal{P}t} y$. It is clear that the rules E5–E9 are sound on this interpretation.

11. Sets $C_{\mathcal{P}t}$ of statements controlled when fresh by $\mathcal{P}$ at time t. We define $\langle w, t\rangle : \mathcal{P}$ **controls** x if and only if $x \in C_{\mathcal{P}t}$. The relation of these sets to the rule C1 will be considered when we discuss the accessibility relation below.

12. Sets $D_{\mathcal{P}t}$ of statements controlled even when not fresh by $\mathcal{P}$ at time t. We define $\langle w, t\rangle : \mathcal{P}$ **controlled** x if and only if $x \in D_{\mathcal{P}t}$.

The semantics of belief is handled in the normal way by defining a transitive (but not reflexive) *accessibility* relation $\rightsquigarrow_{\mathcal{P}}$. We write $w \rightsquigarrow_{\mathcal{P}} w'$ to mean that the possible world w' is consistent with the beliefs of principal $\mathcal{P}$ in world w (i.e. if $\mathcal{P}$ is really in world w, with certain beliefs, then w' is one of the worlds that she thinks she might be in). We require that this relation leaves unchanged $S_{\mathcal{P}t}$, $\hookrightarrow_{\mathcal{P}t}$, $U_{\mathcal{P}t}$ and $\mapsto_{\mathcal{P}t}$ for every t, as well as all pairs $(\mathcal{P}, k)$ in H_t for every t, and all messages to or from $\mathcal{P}$ in M (with the proviso that messages **to** $\mathcal{P}$ in M may have a different sender). We also require that for statements a about the world, if $a \in C_{\mathcal{P}t}$ in w' and (**assert** a) $\in Z_{\mathcal{P}t} \cap F_t$ in w' then a is true about the world w' at time t, i.e. $\langle w', t\rangle : a$. Similarly if $a \in D_{\mathcal{P}t}$ in w' and (**assert** a) $\in Z_{\mathcal{P}t}$ in w' then $\langle w', t\rangle : a$.

We define (in the normal way) $\langle w, t\rangle : \mathcal{P} \models a$ if and only if for every world w' such that $w \rightsquigarrow_{\mathcal{P}} w'$, $\langle w', t\rangle : a$. It is clear that the rules B1–B7 and C1–C2 are sound on this interpretation, and the rules M1–M3 are known to be sound for a semantics of this kind. Soundness of the rules Z6 and Z7 follows from the fact that $x \mapsto_{\mathcal{P}t}$ implies $x \supseteq y$. Note that we can have $\langle w, t\rangle : \mathcal{P} \models$ **false** if there is no world w' such that $w \rightsquigarrow_{\mathcal{P}} w'$.

With the semantics we have chosen, the rules are not complete. In particular, we may not be able to prove $\mathcal{P} \models a$ even if a is true in every accessible world. For example, we may have $x \supseteq y$, but we cannot prove $\mathcal{P} \models x \supseteq y$, so we would use rules Z6 and Z7 which require proving $x \succ\!\mathcal{P}\!\succ y$. We can think of $x \succ\!\mathcal{P}\!\succ y$ as meaning that there is *evidence* to convince $\mathcal{P}$ that

$x \supseteq y$. We could have made the semantics match the rules more closely (by including as a 13th component of the world a relation which is sometimes $\supseteq$ and sometimes $\mapsto_{\mathcal{P}t}$). However, since our goal was to prove soundness, the semantics we have chosen is sufficient. We also feel that the question of whether there is sufficient evidence for a principal to believe something is more easily tackled proof-theoretically, by means of rules such as E1–E9, rather than model-theoretically.

8. Related Work

Prior work in automating protocol objects has followed two approaches differing from ours. Automation in Prolog (e.g. [5]) can provide a useful tool, but since correctness of the results depends on correctness of the complex Prolog program, the results cannot necessarily be treated with total confidence. Automation using a theorem prover like HOL (e.g. [6, 7, 8, 9, 10]) or Isabelle is more promising and has led to useful and powerful tools [7] (although no soundness proof appears to exist for this rule set). However, the use of complex proof tactics can sometimes fail to prove true theorems (especially within belief logic, as discussed in [6]).

In contrast, our approach has been to simplify automation by careful formulation of the rules, in particular the rules X1–X6 and the derived rules D1–D8. With these rules a relatively simple and generic proof tactic can be used while ensuring termination (although, as noted in section 3 above, non-termination is still possible for certain pathological cases). We believe our approach is also more easily generalised to new protocol analysis objects that may be developed in the future.

9. Conclusion

We have presented an improved version of BAN logic which is easily automated and proved sound. We have developed an implementation using Isabelle and a GUI interface. Although we have many more rules than SVO logic (66 rather than 20), the rules are relatively simple (the more complex rules D1–D8 are derived) and hence easier to automate and prove correct. Some of the complexity has been forced by the need to avoid looping in automatic proof tactics, and some by the need to ensure that $\mathcal{P} \models a$ does not have a meaning unrelated to a, since this can lead to unsoundness. We have applied our tool to a number of examples relevant to electronic commerce. The fact that our rules have been proved sound provides us with confidence that the results of our tool are meaningful. This is particularly important because the development of our tool was intended to assist in conducting information security evaluation of cryptographic protocols at the ITSEC E6 level.

However, our logic is not suitable for proving *negative* results (e.g. $\sim \mathcal{E}$ **sees** k, where $\mathcal{E}$ is an eavesdropper). Results of this kind, which are practically of great value, are probably best achieved with a model-theoretic tool, since the absence of a proof does not generally mean that a statement is false.

10. Acknowledgements

The author is deeply indebted to Bernard Colbert, Brendan Mahony, and several anonymous referees for comments on earlier versions of this paper.

References

[1] M. Burrows, M. Abadi and R. Needham, *A Logic of Authentication*, Research Report 39, Digital Systems Research Centre, February 1989

[2] M. Abadi and M. R. Tuttle, *A Semantics for a Logic of Authentication*, Proceedings of the Tenth ACM Symposium on Principles of Distributed Computing, August 1991, 201–216

[3] L. Gong, R. Needham and R. Yahalom, *Reasoning about Belief in Cryptographic Protocols*, Proceedings of the 1990 IEEE Computer Society Symposium on Security and Privacy, 234–248

[4] A. Mathuria, R. Safavi-Naini and P. Nickolas, *Some Remarks on the Logic of Gong, Needham and Yahalom*, Proceedings of the 1994 International Computer Symposium, December 1994, 305-308.

[5] A. Mathuria, R. Safavi-Naini and P. Nickolas, *On the Automation of GNY logic*, Proceedings of the 1995 Australian Computer Conference, January–February 1995, 370-379.

[6] S. Brackin, *An Interface Specification Language for Automatically Analyzing Cryptographic Protocols*, Internet Society Symposium on Network and Distributed System Security, February 1997

[7] R. Lichota, G. Hammonds and S. Brackin, *Verifying Cryptographic Protocols for Electronic Commerce*, Proceedings of the Second USENIX Workshop on Electronic Commerce, November 1996

[8] S. Brackin, *Automated Formal Analyses of Cryptographic Protocols*, Proceedings of the 19th National Conference on Information Systems Security, October 1996

[9] S. Brackin, *A HOL Extension of GNY for Automatically Analyzing Cryptographic Protocols*, Proceedings of Computer Security Foundations Workshop IX, June 1996

[10] S. Brackin, *Deciding Cryptographic Protocol Adequacy with HOL*, Higher Order Logic Theorem Proving and Its Applications, September 1995

[11] P. F. Syverson and P. C. van Oorschot, *On Unifying Some Cryptographic Protocol Logics*, Proceedings of the 1994 IEEE Computer Society Symposium on Security and Privacy, May 1994

[12] K. A. Bowen, *Model Theory for Modal Logic*, D. Reidel, Dordrecht, 1979

[13] M. Fitting, *Proof Methods for Modal and Intuitionistic Logics*, D. Reidel, Dordrecht, 1983

[14] J.-J. Ch. Meyer et al, *Epistemic Logic for Computer Science: A Tutorial (Parts 1 and 2)*, EATCS Bulletin **44** (June 1991, 242–270) and **45** (Oct 1991, 256–287)

[15] L. A. Wallen, *Automated Deduction in Nonclassical Logics*, MIT Press, 1990

Validating the Use of BAN LOGIC

José M. Sierra, Julio C. Hernández, Almudena Alcaide, and Joaquín Torres

Carlos III University of Madrid
Avda. Universidad 30, 28911, Leganés, Madrid, Spain
sierra@inf.uc3m.es

Abstract. Most attacks against security protocols are due to their vulnerable designs. These type of protocols are usually the base which many other protocols and applications are built upon, so proving the correctness of such protocols has become a very important issue in recent years. At the same time, the complexity of security protocols has increased considerably, making it harder to perform an exhaustive analysis of the different situations they are able to deal with. BAN logic was created to assist in the validation of authentication protocols. Although there are other validation logics, we have chosen BAN because we believe its formal process is very simple and robust and therefore facilitates its application to validate old protocols such as Otway-Rees and more complex new ones such as IKE (standard Internet Key Exchange protocol). This paper is based on BAN logic. We will give a brief description of validating procedures and we will demonstrate the validity of BAN foundations, refuting some weaknesses detected by other authors.

1 Introduction

Nowadays, security protocols are widely used, providing security services in different distributed systems. Deficiencies in the design of these protocols could have negative consequences over the system they are supposed to protect. In fact, most protocol attacks try to exploit those design defects instead of attempting against their cryptography elements, which are generally stronger. However, the design of security protocols is not always considered an important task and very often, malicious modifications of protocol messages are not evaluated to ensure that the protocol is still secure. For many years, different authors have been pointing out these type of errors on security protocol designs.

Nowadays, there is quite a heterogeneous set of security protocols differing in the number of participants, the role that each of them plays in the authentication process, the different relationships between them and how these relations develop, etc. For that reason there is the need to create a logical structure to set the bases for the validation of any type of security protocol that could assist in their understanding and avoid potential vulnerabilities.

Burrows, Abadi and Needham made one of the most significant efforts in 1990 defining a logic for the analysis of security protocols [2]. BAN Logic is based on the authentication of entities and how their relationships evolve during the run of a protocol. Furthermore this logic can be used to describe the message exchanging routines without ambiguity, explaining explicitly what assumptions are needed and what information should be considered for the authentication of the participants.

Intentionally, BAN Logic does not consider all aspects of security protocols. This logic operates at an abstract level and therefore does not consider implementation errors or inappropriate use of cryptosystems.

The simplicity of BAN Logic is one of the reasons for its wide use. However, this simplicity means that BAN is not powerful enough to analyse existing security protocols which have features not considered by BAN. Many researchers have tried to solve this problem by redefining BAN. In this way, Needham (one the authors of BAN), Gong and Yahalom, in 1990 introduced a new logic, GNY Logic [5], and a year later, Abadi and Tuttle created [1]. Other authors like Boyd and Mao have reviewed BAN [3] and have created complementary extensions to it such as [4]. Given that Diffie-Hellman protocols underlay most of the modern authenticated-key-distribution protocols, much effort has gone into trying to validate such protocols. Paul van Oorschot´s VO logic [8], and also [6] and [7] were designed primarily to add this capability while retaining various of the BAN foundations.

As shown above, many publications tried to complete and improve BAN logic's features. The majority of these logics were oriented to give a global solution to the validation of security protocols. However, after all these attempts, it seems obvious that the diversity of security protocols make it very difficult to find one single logic which can be used to validate any given security protocol. Consequently, the use of specific extensions of BAN will be a more adaptable solution for the validation of current or future security protocols.

2 What Are BAN Logic Foundations?

BAN logic is a *logic of beliefs*. The intended use of BAN is to analyse authentication protocols by deriving the beliefs that honest principals correctly executing a protocol can come to, as a result of the protocol execution.

Any authentication protocol is based on the exchanging of messages between participants. To validate a protocol using BAN logic we must establish the participants and *their beliefs* at the beginning of such protocol. Also we must be able to express those beliefs using BAN specific notation (see [2] for BAN notation).

Each of the messages exchanged during the run of a protocol is then idealized (this is called the idealization process), i.e., each message is represented by a logical formula using BAN symbols and notation. These formulae are accompanied by a set of assertions, also represented in BAN notation. The assertions express conclusions reached after sending the message.

Roughly, the validating process can be understood in these terms:

[initial beliefs and assumptions]

S1 [assertion 1]

S2 [assertion 2]

...

Sn [conclusions]

where Si are statements sent amongst participants.

Step by step we can follow the evolution from the original assumptions to the conclusions, i.e., from the original beliefs to the final ones.

The goals of authentication vary from one protocol to another. Very often authentication is seeking the distribution of a shared key between participants. In that case authentication is completed between participants A and B, when each of them receives the shared key K_{ab}, which they need to communicate to each other.

In a similar way, using BAN logic terminology, the process of validating a protocol is completed when, from initial beliefs and assumptions, using the assertions given in the process and the inference rules defined in BAN, we can reach the conclusions that principal A has received a key K_{ab} and A believes that Kab is a safe key to communicate with B and, vice versa, principal B has received key K_{ab} and B believes that K_{ab} is a safe key to communicate with A.

Furthermore, BAN establishes some other general assumptions from which we highlight the following ones:

- Each encrypted message contains sufficient redundancy to allow a principal who decrypts it to verify that he has used the right key.
- A message cannot be understood by a principal who does not know the key.
- The idealized protocols do not include cleartext message parts. This is because its contribution to an authentication protocol is mostly providing hints as to what might be placed in encrypted messages. They do not contribute to the beliefs of the recipient, although this does not mean that cleartext could be removed from the real messages.
- The interpretation of the messages is only possible because we know how the information that they contain should be understood.

3 Are BAN Logic Foundations Valid?

Various reviews of BAN logic have tried to demonstrate unsoundness in BAN logic foundations. To illustrate some of these papers, we will work on Boyd and Mao paper [3]. In [3] authors start from protocols proved secure by BAN and establish that they might be vulnerable. The paper begins with the idealization of the Otway-Rees protocol, explicitly described in BAN [2].

Otway and Rees proposed a shared-key authentication protocol which involves two principals and an authentication server. A and B represent the two principals, K_{as} and K_{bs} their private keys and S the authentication server. The principals A and B generate the *nonces*[1] N_a, N_b and M; the server S generates K_{ab} which becomes the session key between A and B. The message sequence is represented in the diagram below (figure 1).

[1] Nonces are expressions generated for the purpose of being *fresh*. They have never been used before the current run of the protocol.

A passes to B some encrypted material useful only to the server, together with enough information for B to make up a similar encrypted message. B forwards both to the server, who decrypts and checks whether the components M, A and B match in the encrypted messages. If so, S generates K_{ab} and embeds it in two encrypted messages, one for each participant, accompanied by the appropriate nonces.

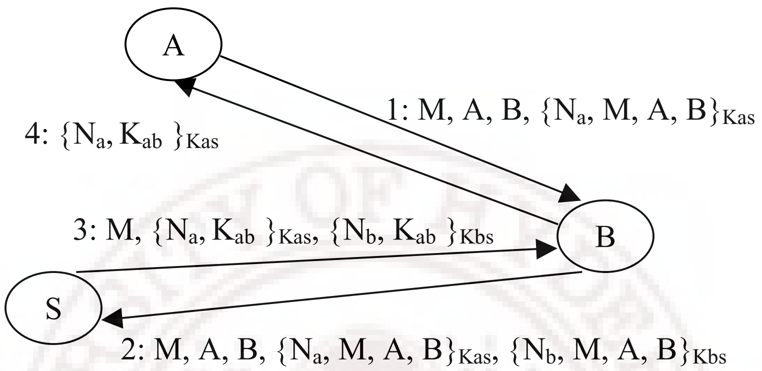

A
B
S
1: M, A, B, {Na, M, A, B}Kas
2: M, A, B, {Na, M, A, B}Kas, {Nb, M, A, B}Kbs
3: M, {Na, Kab }Kas, {Nb, Kab }Kbs
4: {Na, Kab }Kas

Fig. 1. Otaway-Rees Protocol

In order to use BAN logic rules to validate this protocol, we transform the protocol into its idealized version. The nonce N_c corresponds to M, A, B in the protocol description above.

Idealized version of the Otway-Rees protocol:

Message 1: $A \rightarrow B$: $\{N_a, N_c\}_{Kas}$
Message 2: B \title{N3: A Geometrical Approach for Network Intrusion Detection S: $\{N_a, N_c\}_{Kas}, \{N_b, N_c\}_{Kbs}$

Message 3: $S \rightarrow B$: $\{N_a, A \overset{Kab}{\leftrightarrow} B), (B \mid\sim N_c)\}_{Kas}, \{N_b, A \overset{Kab}{\leftrightarrow} B, (A \mid\sim N_c)\}_{Kbs}$

Message 4: $B \rightarrow A$: $\{N_a, A \overset{Kab}{\leftrightarrow} B), (B \mid\sim N_c)\}_{Kas}$

The statements $(A \mid\sim N_c)$ and $(B \mid\sim N_c)$ represent the fact that S has performed the appropriate checks to confirm that N_c matched in each of the encrypted messages. Had this checking not been successful, S would not have issued message 3.

Once the protocol has been idealized, the rest of the procedure consists merely of applying the postulates of the logic and the inference rules to the formulae available. The proof may be briefly outlined as follows:

Initial beliefs and assumptions:

$A \mid\equiv A \overset{Kas}{\leftrightarrow} S$, $S \mid\equiv A \overset{Kas}{\leftrightarrow} S$, $A \mid\equiv (S \Rightarrow A \overset{Kab}{\leftrightarrow} B)$, $A \mid\equiv (S \Rightarrow B \mid\sim X)$,

$B \mid\equiv B \overset{Kbs}{\leftrightarrow} S$, $S \mid\equiv B \overset{Kbs}{\leftrightarrow} S$, $B \mid\equiv (S \Rightarrow A \overset{Kab}{\leftrightarrow} B)$, $B \mid\equiv (S \Rightarrow A \mid\sim X)$,

Kab

$S \models A \leftrightarrow B$

$A \models \#(N_a)$, $A \models \#(N_c)$, $B \models \#(N_b)$,

After message 1 has been sent, B sees the message but does not understand it:
$B \triangleleft \{N_a, N_c\}_{Kas}$

B is able to generate a message of the same form and to pass it on to S along with A´s message. On receiving message 2, S decrypts each encrypted message and checks that the nonce N_c matches in both. Using BAN postulates the following formulae can be inferred prior message 3 is sent:

$S \models A \mid\sim N_a$, $S \models A \mid\sim N_c$,
$S \models B \mid\sim N_b$ and $S \models B \mid\sim N_c$

S emits a message containing two encrypted parts, one for B containing K_{ab} and N_b, the other one, containing the key K_{ab} and N_a, is intended for A, so B has to pass it on.

At this point, both A and B have received a message from the server containing a new encryption key and the nonce they generated in the request messages. Then the following final beliefs emerge:

$A \models A \overset{Kab}{\leftrightarrow} B$, $A \models B \mid\sim N_c$

$B \models A \overset{Kab}{\leftrightarrow} B$, $B \models A \mid\sim N_c$

According to the Boyd and Mao document [3], another entity *T* could impersonate B and send the following message 2'to S; S would then respond to *T* creating the following situation:

Message **2'**: $\boldsymbol{T} \rightarrow S$: $M, A, T, \{N_a, M, A, B\}_{Kas}, \{N_t, \mathbf{M, A, B}\}_{Kts}$
Message **3'**: $S \rightarrow \boldsymbol{T}$: $\{N_a, K_{at}\}_{Kas}, \{N_t, K_{at})\}_{Kts}$
Message **4'**: $\boldsymbol{T} \rightarrow A$: $\{N_a, K_{at}\}_{Kas}$

The attack intends to confuse S including an encrypted message using K_{ts} when *T* is not one of the original participants. If the attack is successful then A would believe that the key it receives is a shared key to communicate with B, when in fact is a key it shares with the attacker *T*.

It is true that in [2] the authors do not explicitly say that the server S must check if the shared keys belong to the initial participants of the communication. However, BAN notation does implicitly indicate that this check is performed and the following belief is inferred from that action:

$$S \models \mathbf{B} \mid\sim N_c$$

So, in our opinion, this is not a vulnerability of BAN formal process but only a reasonable assumption which BAN´s authors do not explicitly mention.

Moreover, in the same paper [3], Boyd and Mao describe a different attack carried out on a simplified version of the Otway-Rees protocol. In [2], the authors conclude that the protocol created by Otway and Rees is valid but with various forms of redundancy so they propose a simplified version to it:

"*Two nonces are generated by A; however the verification using N_a could just as well have been done using N_{c}. Therefore, N_a can be eliminated, so reducing the amount of encryption in the protocol. Moreover, it is clear from the analysis that N_b need not be encrypted in the second message. As these possibilities are explored, we rapidly move towards an improved protocol of different structure*" [2].

In our opinion, after the changes introduced to simplify the protocol, the logical analysis of it, is obviously different from the logical analysis to validate the original one. We will now demonstrate how by adding the necessary assumptions to the set of initial beliefs, BAN logic is able to validate the simplified protocol and the attack described by Boyd and Mao is actually detected by the participants.

According to BAN authors, the new protocol is the same as the Otway-Rees protocol but the messages 1 and 2 are now as follows:

Message 1: $A \rightarrow B$: $M, A, B, \{ N_c \}_{Kas}$
Message 2: $B \rightarrow S$: $M, A, B, \{ N_c \}_{Kas}, N_b, \{ N_c \}_{Kbs}$

The idealized form of the simplified protocol is:
Message 1: $A \rightarrow B$: $\{ N_c \}_{Kas}$
Message 2: $B \rightarrow S$: $\{ N_c \}_{Kas}, \{ N_c \}_{Kbs}$

Message 3: $S \rightarrow B$: $\{N_c, A \overset{Kab}{\leftrightarrow} B), (B \mid\sim N_c)\}_{Kas}, \{N_b, A \overset{Kab}{\leftrightarrow} B, (A \mid\sim N_c)\}_{Kbs}$

Message 4: $B \rightarrow A$: $\{N_c, A \overset{Kab}{\leftrightarrow} B), (B \mid\sim N_c)\}_{Kas}$

To validate this new protocol using BAN, we must add the appropriate *assertions* after each message is sent. After message 1 is sent, the following BAN logic formula can be added to the validating process:

$B \triangleleft \{N_c \}_{Kas}$

After message 2 is sent, S has to perform the required checks. If S succeeds, then we can derive the following beliefs, applying BAN inference rules:

$$S \models A \mid\sim N_c \text{ and } S \models B \mid\sim N_c$$

However, we cannot conclude that $S \models B \mid\sim N_b$. Therefore, new assumptions are needed to complete the authentication process of the participant B. Different solutions can be applied to solve the problem. We briefly describe two of them.

- S checks that N_c does not correspond to any previous run of the protocol. If this is the case, the formula $S \models \# N_c$ could be added, after message 2, to the validating process.
- S checks that K_{ab} has not been requested for any of the participants before this run of the protocol. In this case the formula added to the process should be $S \not\models \# K_{ab}$, this is, S has never said K_{ab} before the current run of this protocol.

According to Boyd and Mao [3], in the new attack, the attacker *T* masquerades as A in the protocol and is also assumed to be in control of communications between B and the server S. The essence of the attack is that *T* can change the names presented to S while using the nonce that B associates with K_{ab}. It is also assumed that *T* has possession of a message fragment $\{M,T,B,\}K_{bs}$, which was formed by B during a previous legitimate run of the protocol between *T* and B. The attack proceeds as follows, with B and S acting exactly as in a normal run. Messages 2 and 3, which B and S intend for each other respectively, are captures by *T*:

Message 1: $T \rightarrow B$: M', A, B, $\{ M, T, B \}_{Kts}$
Message 2: $B \rightarrow T$: M', A, B, $\{ M, T, B \}_{Kts}$, N_b, $\{ M', A, B \}_{Kbs}$
Message 2': $T \rightarrow S$: M, *T*, B, $\{ M, T, B \}_{Kts}$, N_b, $\{ M, T, B \}_{Kbs}$
Message 3: $S \rightarrow T$: $\{M, K_{tb} \}_{Kts}$, $\{ N_b, K_{tb}\}_{Kbs}$
Message 3': $T \rightarrow B$: $\{M, K_{tb} \}_{Kts}$, $\{ N_b, K_{tb}\}_{Kbs}$
Message 4: $B \rightarrow T$: $\{M, K_{tb} \}_{Kts}$

At the end of this attack, B believes he shares the key K_{tb} with A whereas in fact, it shares it with attacker *T*.

We can see that any of the solutions given to perform BAN logic validation of this new protocol, could detect such attack. Both solutions ensure that messages from previous runs cannot be used in future ones. With the first solution, S can detect the attack when realising that N_c = *M, T*, B is not fresh, i.e., it has already been used in some previous instance of the protocol. Also, the assumption added in the second solution ensures that if the encrypted message $\{ M, T, B \}_{Kbs}$ had already been created by B, during a previous legitimate run of the protocol between T and B, then the key K_{tb} would have been issued once already, so such key is not fresh and S can then detect the attack.

4 Conclusions

We believe BAN logic foundations are valid. BAN logic represents a simple but sound and powerful tool to describe and validate authentication protocols. However we are also aware of the limitations of BAN's initial versions. In these early versions the idealization process is strongly based upon the previous understanding of the content of each message exchanged and there are difficulties trying to idealize Diffie-Hellman and other more modern and complex protocols. A well known example is that BAN logic is unable to evaluate protocols where the value of the postulates changes during the run of the protocol (i.e. many e-commerce protocols).

It is vital to validate security protocols to protect communication over open environments such as the Internet. Lack of attention to mutual authentication, freshness of the message exchange, privacy of classified information or impersonation of entities are the main problems associated with these protocols.

Although there are other methods to detect vulnerabilities in these types of protocol, formal validation has become one of the most convenient solutions.

In this paper we have proved the validity of BAN foundations refuting some weaknesses detected by other authors.

Our specific project is to now build *dedicated logical structures* based on BAN, to be used as '*scaffolding*' to validate new security protocols on different new platforms.

References

[1] Martín Abadi and Mark R. Tuttle. A semantic for a logic of authentication. Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing, pages 201-216. ACM Press, August 1991.

[2] Michael Burrows, Martín Abadi and Roger Needham. A logic of authentication. ACM Transactions on Computer Systems, 8(1):18-36, Feb 1990.

[3] Colin Boyd and Wenbo Mao. On a limitation of BAN logic. Eurocrypt'93. Protocols I:240-247, May 1993.

[4] Colin Boyd. A Framework for Design of Key Establishment Protocols. Information Security and Privacy, LNCS 1172, pp.146-157, Springer-Verlag, 1996.

[5] Li Gong, Roger Needham and Raphael Yahalom. Reasoning about Belief in Cryptographic Protocols. Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy, pages 234-248. IEEE Computer Society Press, 1990.

[6] Paul F. Syverson and Paul C. van Oorschot. On unifying some cryptographic protocols. Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy, pages 14-28. IEEE CS Press, May 1994.

[7] Paul F. Syverson and Paul C. van Oorschot. A Unified Cryptographic Protocol Logic. NRL Publication 5540-227, Naval Research Lab, 1996.

[8] Paul C. van Oorschot. Extending Cryptographic logics of belief to key agreement protocols. In Proceedings of the 1st ACM Conference on Computer and Communications Security, pages 233-243. ACM Press, November 1993.

Inf Syst Front (2008) 10:19–32
DOI 10.1007/s10796-007-9049-0

Formalising theories of trust for authentication protocols

Ji Ma · Mehmet A. Orgun

Published online: 4 October 2007

Abstract This paper discusses a formal approach for establishing theories of trust for authentication systems which can be used to reason about how agent beliefs evolve through time. The goal of an authentication system is to verify and authorise users in order to protect restricted data and information, so trust is a critical issue for authentication systems. After authentication, two principals (people, computers, services) should be entitled to believe that they are communicating with each other and not with intruders. So, it is important to express such beliefs precisely and to capture the reasoning that leads to them. In this paper, we focus on analysis of agent beliefs in dynamic environments using a temporalised belief logic, obtained by adding a temporal logic onto a belief logic. Working through a well-known authentication protocol, namely Kerberos, we discuss how to express principal beliefs involved in authentication protocols and the evolution of those beliefs based on a series of observations of agents as a consequence of communication. Our approach could be used for designing, verifying and implementing authentication protocols.

Keywords Agents · Authentication protocols · Belief logic · Trust theory

J. Ma (✉) · M. A. Orgun
Department of Computing, Macquarie University, Sydney, NSW 2109, Australia
e-mail: jma@ics.mq.edu.au

M. A. Orgun
e-mail: mehmet@ics.mq.edu.au

1 Introduction

1.1 Motivation

Authentication protocols are the basis of security in many distributed systems (Burrows et al. 1990; Needham and Schroeder 1978). The goal of an authentication protocol is to authenticate users. After authentication, two principals (people, computers, services) should be entitled to believe that they are communicating with each other and not with intruders. So, it is important to express such beliefs precisely and to capture the reasoning that leads to them. Therefore trust is a critical issue for authentication systems. Trust influences not only the specification of authentication protocols, but also the techniques needed to implement and manage such protocols.

An authentication protocol can be formalised as a kind of trust theory, which can be used to reason about the trust of agents. Reasoning about trust actually involves reasoning about beliefs, therefore a theory of trust may need to be based on a kind of belief logic. In a trust model proposed by Liu and Ozols (2002), it is assumed that for security considerations initially principals (agents) may not trust anyone but the security mechanisms (as special agents) of a system whose trustworthiness has been verified based on required evaluation criteria. Thus, the beliefs of agents can be obtained based on their initial trust in the security mechanisms of the system. The initial trust or meta-beliefs of agents in the system can be encapsulated in a notion of trust and represented as a set of rules (axioms) in a chosen logical framework. These rules together with those of the logic form a theory of trust for the system (Liu 2001). Such theories provide a foundation for reasoning about agent beliefs as well as security properties that systems may satisfy.

On the other hand, since communication protocols operate in dynamic, unpredictable environments, it is also essential to model the evolution of a system through time. In particular, we need to be able to express the initial principal beliefs involved in authentication protocols and the evolution of these beliefs as a consequence of communication. The motivation of this work is to provide a formal approach for establishing theories of trust for authentication systems which can be used to reason about agent beliefs which may vary through time.

1.2 Related work

There are many studies discussing how to use a kind of belief logic to capture the notion of trust and to describe authentication protocols, but only a few studies that focus on dynamic theories of trust for authentication systems. Burrows et al. (1990) proposed a logic called BAN to describe the beliefs of trustworthy parties involved in authentication protocols and the evolution of these beliefs. Many researchers have developed extensions of the BAN logic to overcome it's deficiencies by generalisation of its assumptions (Gong et al. 1990; Syverson and Oorschot 1996; Oorschot 1993).

The GNY belief logic (Gong et al. 1990) extends the BAN logic to require fewer assumptions and offers a few extra features. The SvO logic (Syverson and Oorschot 1996) corrects flaws found in the BAN logic and other similar logics; SvO logic is easier to use and more expressive than the BAN logic. Campbell et al. (1992) extend the BAN logic to support probabilistic reasoning for calculating a measure of trust. A recent work proposed by Wen et al. (2005) discusses the functions of the BAN logic in formalization analysis and explores its limitations, and suggests methods for improvement. Some other belief logics include: Moser (1989) logic for reasoning about beliefs of protocol participants and trust changes, Wedel and Kessler (1996) logic used for the analysis of authentication protocols and its soundness, and the Yahalom et al. (1993) system that can be applied to analyse the nature of the trust that protocol participants must have in each other.

As mentioned earlier, trust changes dynamically; once agents lose their trust in the security mechanisms of a system, the theory based on the initial trust of the system is no longer valid, so it must be revised or not be used for any security purpose (Liu et al. 2004). Regarding belief revision, Jonker and Treur (1999) propose two functions, trust evolution function and trust update function; Gabbay et al. (2003) proposed a Controlled Revision Approach. Generalizations of theory revision have become known as base change (Meyer 1999; Schulte 1999). Theories of trust discussed in this paper can also be regarded as theory bases. Such a theory is a basic set$\check{T}$ describing meta-beliefs of agents, therefore revising a theory of trust is in fact the revision of the base of the theory. With the base change, revising a theory can usually be conducted by defining appropriate base change operations. There are some more recent papers (Giacomo et al. 2006; Liu et al. 2006) that discuss formal update semantics for description logics.

However, most of those papers have not discussed modelling and reasoning about the evolution of agent beliefs through time. Furthermore they have not discussed how to revise a theory in dynamic environments. In our previous work (Ma and Orgun 2006b) we have proposed a general approach for trust management within agent systems. As the approach does not include a temporal component, it can not deal with the evolution of agent beliefs. In this work we discuss a methodology for establishing theories of trust for authentication systems using a temporal belief logic, and provide a method to revise such theories in dynamic environments. This paper extends our recent works on the specification of agent beliefs (Ma and Orgun 2006a, 2007).

1.3 Aims and contributions

A basic problem regarding security properties of an authentication system is whether a message sent by a principal through the system is reliable in the view of receivers. The problem generally depends on the security mechanisms of a system and the trust that agents would put in the mechanisms. Therefore, it is important to provide a formal method for specifying the trust that agents have in the security mechanisms of the system; the beliefs of agents can be represented as a set of rules, and these rules form a theory of trust for the system.

In this paper, we use a temporalised belief logic called TML$^+$ (Liu et al. 2004) to formalise theories of trust for authentication systems. TML$^+$ is obtained by combining a belief logic called TML (typed modal logic; Liu 2001) with a temporal logic called SLTL (simple linear-time temporal logic). SLTL is a simple form of temporal logic with clocks proposed by Liu and Orgun (1996, 1999) for specifying reactive systems in which different events may occur over timelines with varying rates of progress. This logic allows us to express agent beliefs as temporal propositions that may vary through time.

Focusing on the dynamics of trust, we first show how to establish a theory of trust for authentication systems in TML$^+$ and then discuss how to specify trust changes and revise such theories based on dynamic changes of trust once they have been established.

The main contributions of this paper are as follows:

- Providing a method for establishing a theory of trust for authentication systems, which includes a technique for expressing agent beliefs within authentication systems.
- Providing a method for modelling the dynamics of trust within authentication systems, which includes a method for expressing a trust state and a technique for expressing trust changes to a given trust state.
- Providing a general method for revising theories of trust for authentication.

The rest of this paper is organised as follows: Section 2 briefly introduces the syntax. semantics and inference rules of the logic TML$^+$ on which our approach is based. Section 3 presents an approach for constructing theories of trust for authentication systems in TML$^+$ using Kerberos authentication system as an example. Section 4 discusses modelling trust change and trust theory revision. Section 5 concludes the paper.

2 Temporalised belief logic TML$^+$

In this section, we discuss the reason why the logic TML$^+$ is chosen to express and reason about agent beliefs and build trust theories. A motivating example is also given to demonstrate how it works.

2.1 Motivation for TML$^+$

Belief represents a disposition of an agent to a proposition, so a logic of expressing propositional dispositions should be able to specify the required relations between believers and attitudes (Rangan 1988). Classical first-order logic cannot handle such relations well. The modal logic approach is able to enhance propositional and first-order logic with modal operators to represent agent beliefs. Liu (2001) proposed a belief logic, called TML (typed modal logic), which extends first-order logic with typed variables and belief operators. TML is very suitable to express static properties, for example, the assertion "John believes that Bob has the key k" can be formalised in TML as $\mathbf{B}_{\text{john}}$ Has(bob, k) where $\mathbf{B}_{\text{john}}$ is the modal belief operator for agent John. However, trust is the outcome of observations leading to the belief that the actions of another may be relied upon, without explicit guarantee, to achieve a goal in a risky situation. Trust changes or evolves dynamically (as time progresses), when we gain confidence or lose confidence in the security mechanisms of the system. An agent may be trusted by some agents today, but tomorrow the situation may change. Suppose that we have the assertion that "John believes that Bob has the key today but tomorrow John believes that Bob does not have the key." Without the introduction of a temporal dimension, TML may not be able to express such dynamics.

The purpose of this work is to build a logical framework, in which we are able to describe both the *statics* and *dynamics* properties of trust. Temporal logics have the ability to deal with dynamics, and they have been used successfully in many areas, such as specifying, modelling and specifying communication protocols. In a later work, Liu et al. (2004) used the temporalising technique proposed by Finger and Gabbay (1992) to add a temporal dimension to TML, by combining it with the simple linear-time temporal logic (SLTL). The resulting logic, TML$^+$, can be applied for reasoning about time-dependent properties regarding agent beliefs in multi-agent systems. In particular, it has a strong expressive power for reasoning about time-dependent properties regarding agent beliefs for communication systems and in the analysis of dynamic security properties.

TML$^+$ has two classes of modal operators: (1) belief operators; and (2) temporal operators. The belief operators are those of TML whereas the temporal operators are those of SLTL. Each agent has an associated belief operator, for instance, the belief operator, $\mathbf{B}_{Bob}$, is intended to denote "Bob believes that". The temporal operators contained in TML$^+$ are **first** and **next**, which refer to the initial moment and the next moment in time respectively. In TML$^+$, belief operators and temporal operators are applied to formulas, but not to terms of the language. Since the logic TML$^+$ is obtained by using temporalisation, which is a hierarchy combination technique for combining logics, the temporal operators **first** and **next** can never appear within the scope of a belief operator $\mathbf{B}_i$ or quantifiers $\forall$ and $\exists$.

Below we give a formalization of the *Three Wise Man* puzzle to demonstrate how TML$^+$ can be used to specify and reason about evolving agent beliefs. The following scenario of the puzzle was originally introduced by McCarthy (1990):

> A certain king wishes to determine which of his three wise men is the wisest. He arranges them in a circle so that they can see and hear each other and tells them that he will put a white or black

> spot on each of their foreheads but that at least one spot will be white. In fact all three spots are white. He then offers his favor to the one who will first tell him the color of his spot. After a while, the wisest announces that his spot is white. How does he know?

We follow the assumptions made in the solution by Fisher and Ghidini (2002) based on a temporal belief logic. Each wise man in the puzzle is represented by an agent of TML$^+$ (say $wm1$, $wm2$, $wm3$). Agents may know about beliefs of other agents and they may also have some common knowledge. At the first moment in time (time 0), the king asks the question to all the agents but none of the agents know the answer. At the next moment in time (time 1), the king asks the question to all the agents but then again none of the agents know the answer. At the next moment in time (time 2), however, all the agents know the answer. Here is the formalization of the puzzle and the additional assumptions. Note that these formulas stand for all their instantiations where v and w are substituted for values in the agent domain $\{wm1, wm2, wm3\}$.

W1. **first** HasWhiteSpot(v)
W2. HasWhiteSpot(v) $\rightarrow$ **next** HasWhiteSpot(v)
W3. $\neg$ HasWhiteSpot(v) $\rightarrow$ **next** $\neg$ HasWhiteSpot(v)
W4. HasWhiteSpot(v) $\rightarrow$ $\mathbf{B}_w$ HasWhiteSpot(v) where $v \neq w$
W5. $\neg$ HasWhiteSpot(v) $\rightarrow$ $\mathbf{B}_w$ $\neg$ HasWhiteSpot(v) where $v \neq w$
W6. HasWhiteSpot($wm1$) $\vee$ HasWhiteSpot($wm2$) $\vee$ HasWhiteSpot($wm3$)
W7. **first** $\neg$ $\mathbf{B}_v$ HasWhiteSpot(v)
W8. **first next** $\neg$ $\mathbf{B}_v$ HasWhiteSpot(v)
W9. **first next next** $\mathbf{B}_v$ HasWhiteSpot(v)

These formulas together form a theory for the *Three Wise Man* puzzle. W1 says initially all the agents have a white spot in their heads. W1 is not a statement to be believed by any agent because belief operators cannot be applied to formulas with temporal operators in them, whereas W6 is common knowledge (that is, at least one agent has a white spot on his head). W2–W3 say that whitespots persist; W4–W5 formalize agents beliefs about whitespots on the heads of other agents. W7–W9 formalize the answers of the agents to the king's question at times 0, 1 and 2.

2.2 Background on SLTL and TML

This section discusses SLTL and TML for completeness. SLTL is a discrete linear-time logic where the underlying collection of time points is the set of natural numbers with its usual ordering relation $<$. A time model for the logic SLTL has the form $\mathcal{M}_{sltl} = \langle CK, <, \pi \rangle$, where $CK = \langle t_0, t_1, t_2, \ldots \rangle$ is a clock, $<$ is the usual ordering relation over CK, and π is an assignment function that gives a value $\pi(t, p) \in \{true, false\}$ for any time point t in CK and any atomic formula p. The meaning of SLTL formulas are defined with respect to given local clocks (subsequences of a global clock). The global clock is the increasing sequence of natural numbers, i.e., $\langle 0, 1, 2, \ldots \rangle$ and a local clock is defined as an infinite subsequence of the global clock.

In a simplified form, the semantics of the temporal operators of SLTL are given as follows (Where $CK = \langle t_0, t_1, t_2, t_3, \ldots \rangle$ is a local clock):

- $\mathcal{M}_{sltl}, t_i \models$ **first** φ iff $\mathcal{M}_{sltl}, t_0 \models \varphi$.
- $\mathcal{M}_{sltl}, t_i \models$ **next** φ iff $\mathcal{M}_{sltl}, t_{i+1} \models \varphi$.

Table 1 gives an explanation of the interpretation of the temporal operators of SLTL.

In Table 1, suppose that φ is the formula Has(bob, k) and $ck = \langle t_0, t_1, t_2, t_3, t_4, t_5, t_6, t_7, \ldots \rangle$ is a given local clock. Suppose that the meaning of formula φ over the clock ck is as given as in the first line of Table 1 (where φ is true at times t_0 and t_1 and false at times t_2 and t_3 and so on). Since the initial time of ck is t_0, the meaning of a formula of the form **first** φ at any given moment in time is defined by the meaning of φ at time t_0 (e.g., true in the above example). The meaning of a formula of the form **next** φ at any given moment in time t_i is defined by the meaning of φ at time t_{i+1}; for example, at time t_3 φ is false, but **next** φ is true because φ is true at time t_4.

TML is a variant of the modal logic **KD** of beliefs (Hughes and Cresswell 1996). Let us assume that there are n agents and there are n corresponding modal operators $\mathbf{B}_1, \ldots, \mathbf{B}_n$ in the logic, where $\mathbf{B}_i$ $(1 \leq i \leq n)$ stands for "agent i believes that". A *classical Kripke model* (Kripke 1963) for TML is defined as a tuple $\mathcal{M}_{tml} = \langle S, R_1, \ldots, R_n, \pi \rangle$, where S is the set of states or possible worlds; and each R_i, $i = 1, \ldots, n$, is a relation over S, (called the *possibility relation* according to agent i). R_i is a non-empty set consisting of state pairs (s, t) such that $(s, t) \in R_i$ iff, at state s, agent i

Table 1 Interpretation of temporal operators on a given clock (CK). Here φ is a formula; T represents value **true** and F represents value **false**

Formula	Truth value								
φ	T	T	F	F	T	F	T	F	…
first φ	T	T	T	T	T	T	T	T	…
next φ	T	F	F	T	F	T	F	…	…
CK	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	…

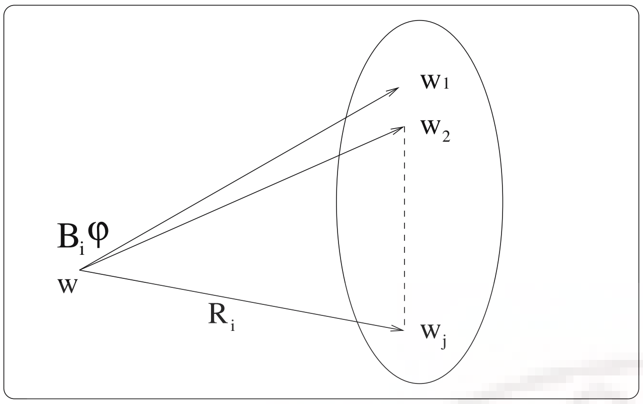


Fig. 1 Interpretation of belief operators (R_i is the possibility relation for agent i; $w_1, \ldots, w_j$ are the states in S that are accessible from w)

considers the state t possible (or accessible); and π is the *assignment function*, which gives a value $\pi(s, q) \in \{true, false\}$ for any $s \in S$ and atomic formula q. Formula φ is satisfiable in the model $\mathscr{M}_{tml}$ if there exists $s \in S$ such that $\mathscr{M}_{tml}, s \models \varphi$. The semantics of the belief operators of TML is given as follows:

- $\mathscr{M}_{tml}, s \models \mathbf{B}_i\varphi$, iff for all t such that $(s, t) \in R_i$, $\mathscr{M}_{tml}, t \models \varphi$.

Figure 1 gives an explanation of the interpretation of belief operators of TML. Recall that R_i is the *possibility relation* according to agent i. In other words, for a given state w, agent i considers all the states $w_1, w_2, \ldots, w_j$ possible. Then a formula of the form $\mathbf{B}_i\,\varphi$ is true at a given state w if and only if φ is true at all states accessible from w with respect to the relation R_i (state w may or may not be accessible from w itself).

When SLTL is *added* to TML using the method of Finger and Gabbay (1992), substitutions of formulas of TML for the atoms of SLTL are allowed, but substitutions of formulas of SLTL atoms of TML are not allowed. TML$^+$ is constructed by adding SLTL onto TML in this way. In the resulting logic, there are certain restrictions on the use of temporal and belief operators, because of the hierarchical combination of belief and temporal logics used. Hence in TML$^+$, we can express statements about temporal aspects of agent beliefs, such as "At the initial moment in time, John believes that Bob has the key k": **first** $\mathbf{B}_{john}$ Has(bob, k). But we cannot express a statement asserting that some agent believes an event to happen at some time, such as "John believes that at the initial moment in time Bob has the key k", that is, we do not have the formula $\mathbf{B}_{john}$ **first** Has(bob, k) in TML$^+$.

2.3 Syntax, semantics and inference rules of TML$^+$

A detailed discussion of syntax and semantics and inference rules of TML$^+$ is given below. For combining TML and SLTL, let $\mathscr{L}_{tml}$ represent the set of all formulas in TML. We divide $\mathscr{L}_{tml}$ into two sets, $\mathscr{L}^{(b)}_{tml}$ and $\mathscr{L}^{(m)}_{tml}$. The set $\mathscr{L}^{(b)}_{tml}$ includes all formulas with Boolean connectives; and monolithic formulas belong to $\mathscr{L}^{(m)}_{tml}$. The set $\mathscr{L}^{(m)}_{tml}$ is defined as follows:

1. $\varphi \in \mathscr{L}^{(m)}_{tml}$, if φ is an atomic formula in $\mathscr{L}_{tml}$.
2. $\mathbf{B}_i\,\varphi \in \mathscr{L}^{(m)}_{tml}$, for all i $(1 \le i \le n)$, if $\varphi \in \mathscr{L}^{(m)}_{tml}$.
3. $\forall X \varphi(X) \in \mathscr{L}^{(m)}_{tml}$, if $\varphi(X) \in \mathscr{L}^{(m)}_{tml}$ and X is any variable.

Let $\mathscr{L}_{tml^+}$ be the set of formulas of TML$^+$. Then $\mathscr{L}_{tml^+}$ is defined as follows:

1. If $\varphi \in \mathscr{L}^{(m)}_{tml}$, then $\varphi \in \mathscr{L}_{tml^+}$.
2. If $\varphi \in \mathscr{L}_{tml^+}$ and $\psi \in \mathscr{L}_{tml^+}$, then $\neg\varphi \in \mathscr{L}_{tml^+}$ and $(\varphi \wedge \psi) \in \mathscr{L}_{tml^+}$.
3. If $\varphi \in \mathscr{L}_{tml^+}$, then **first** $\varphi \in \mathscr{L}_{tml^+}$ and **next** $\varphi \in \mathscr{L}_{tml^+}$.

To be able to interpret a formula of TML$^+$ whose main operator is a temporal operator, we need to use the meaning of the temporal operators with a time reference. To be able to interpret a formula whose main operator is a belief operator, similarly, we need to use the meaning of the belief operators with a state reference. This will require us to move from time references to state references freely, but in one direction only because temporal operators are never within the scope of a belief operator. The temporalisation method combines the semantics of the constituent logics using a mapping that associates each moment in time with a classical Kripke model in such a way that any formula of TML$^+$ is interpreted in its proper context. The meaning definition of any formula of TML$^+$ would involve a time model and a time reference initially, however, as soon as a belief operator is encountered, the current time reference would be mapped to the classical Kripke model under which the meaning of the subformula (to which the belief operator was applied) would be decided. Details are given below.

Let $\mathscr{K}^+$ be a class of (Kripke) models of the logic TML of the form $\mathscr{M}_{tml} = \langle S, R_1, \ldots, R_n, \pi, s\rangle$, where s is the current world, from which the observation is made (Finger and Gabbay 1992). Consider a time frame $(CK, <)$, where $CK = \langle t_0, t_1, t_2, \ldots\rangle$, and a function $v : CK \to \mathscr{K}^+$, mapping moments in time on the clock CK to a model in the class $\mathscr{K}^+$. A model of TML$^+$ is a triple

$\langle CK, <, v\rangle$, denoted by $\mathscr{M}_{tml^+}$. The semantics of TML$^+$ formulas are given as follows :

1. $\mathscr{M}_{tml^+}, t_i \models \varphi, \varphi \in \mathscr{L}^{(m)}_{tml}$, iff $v(t_i) = \mathscr{M}_{tml}, \mathscr{M}_{tml} \models \varphi$.
2. $\mathscr{M}_{tml^+}, t_i \models \neg\varphi$, iff $\mathscr{M}_{tml^+}, t_i \not\models \varphi$.
3. $\mathscr{M}_{tml^+}, t_i \models \varphi \wedge \psi$ iff $\mathscr{M}_{tml^+}, t_i \models \varphi$ and $\mathscr{M}_{tml^+}, t_i \models \phi$.
4. $\mathscr{M}_{tml^+}, t_i \models \textbf{first}\ \varphi$ iff $\mathscr{M}_{tml^+}, t_0 \models \varphi$.
5. $\mathscr{M}_{tml^+}, t_i \models \textbf{next}\ \varphi$ iff $\mathscr{M}_{tml^+}, t_{i+1} \models \varphi$.

We assume the standard definitions of logical connectives such as $\vee$ (or), $\rightarrow$ (implication) and $\leftrightarrow$ (biconditional). The semantics of quantifiers $\forall, \exists$ are defined in the standard way with respect to TML models based on rigid domains for terms (Fitting and Mendelsohn 1999).

The axiom set of TML$^+$ consists of the following axiom schemata. Let $\bigtriangledown$ stand for a temporal or a belief operator.

A0. all axioms of classical first-order logic, including substitution.
A1. $\bigtriangledown(\varphi \rightarrow \psi) \leftrightarrow (\bigtriangledown\varphi \rightarrow \bigtriangledown\psi)$.
A2. $\bigtriangledown(\varphi \wedge \psi) \leftrightarrow (\bigtriangledown\varphi) \wedge (\bigtriangledown\psi)$.
A3. $\forall X(\mathbf{B}_i\, \varphi(X)) \rightarrow \mathbf{B}_i\, (\forall X\varphi(X))$, for all i $(1 \leq i \leq n)$.
A4. $\mathbf{B}_i(\neg\varphi) \rightarrow \neg(\mathbf{B}_i\, \varphi)$.
A5. $\textbf{first}(\neg\varphi) \leftrightarrow \neg(\textbf{first}\ \varphi)$.
A6. $\textbf{next}(\neg\varphi) \leftrightarrow \neg(\textbf{next}\ \varphi)$.
A7. $\textbf{first}(\textbf{first}\ \varphi) \leftrightarrow \textbf{first}\ \varphi$.
A8. $\textbf{next}(\textbf{first}\ \varphi) \leftrightarrow \textbf{first}\ \varphi$.

The inference rules of the logic TML$^+$ include:

IR1. $\dfrac{\varphi,\ \varphi \rightarrow \psi}{\psi}$ (modus ponens)

IR2. $\dfrac{\forall X\varphi(X)}{\varphi(Y)}$ (Instantiation)

IR3. $\dfrac{\varphi(X)}{\forall X\varphi(X)}$, only if φ(X) does not have any temporal operator. (Generalisation)

IR4. $\dfrac{\varphi}{\textbf{first}\ \varphi}, \dfrac{\varphi}{\textbf{next}\ \varphi}$ (Temporal Necessitation)

IR5. $\dfrac{\varphi}{\mathbf{B}_i\, \varphi}$, only if φ does not have any temporal operator. (Belief Necessitation)

IR6. $\dfrac{\varphi,\ \psi}{\varphi \wedge \psi}$ ($\wedge$_introduction)

IR7. $\dfrac{\varphi \wedge \psi}{\varphi}, \dfrac{\varphi \wedge \psi}{\psi}$ ($\wedge$_elimination)

IR8. $\dfrac{\varphi,\ \varphi \leftrightarrow \psi}{\psi}, \dfrac{\psi,\ \varphi \leftrightarrow \psi}{\varphi}$ ($\leftrightarrow$_elimination)

The soundness for the logic TML$^+$ depends on the soundness theorems for TML and SLTL. For the details of the completeness of the logic TML$^+$, we refer readers to Liu et al. (2004).

2.4 Reasoning about agent beliefs with TML$^+$

The main purpose of an authentication system is to verify users. For reasoning about initial agent beliefs within an authentication system we define the following predicates:

- Send(p, msg): Message *msg* was sent by the principal p.
- Receive(p, msg): The principal p has received message *msg*.
- Shared(p, q, k): The principals p and q use the shared key k to communicate.
- Owner(p, k): The principal p has key k as a secret key.
- Fresh(msg): The message *msg* is fresh.
- Reliable(msg): The message *msg* is reliable.

Now we discuss the main logical postulates that we can use in proofs within authentication systems. We postulate that (here all variables are assumed to be universally quantified with respect to their types):

P0. $\mathbf{B}_p$ Shared(p, q, k) $\wedge$ Receive$(p, \{msg\}k)$ $\rightarrow$ $\mathbf{B}_p$ Send(q, msg).
P1. $\mathbf{B}_p$ Owner(q, k) $\wedge$ Receive$(p, \{msg\}k)$ $\rightarrow$ $\mathbf{B}_p$ Send(q, msg).
P2. $\mathbf{B}_p$ Owner(p, k) $\wedge$ Receive$(p, \{msg\}k)$ $\rightarrow$ $\mathbf{B}_p$ Reliable(msg).
P3. $\mathbf{B}_p$ Send(q, msg) $\wedge$ $\mathbf{B}_p$ Fresh(msg) $\rightarrow$ $\mathbf{B}_p\, \mathbf{B}_q$ Reliable(msg).
P4. $\mathbf{B}_p$ Send(q, msg) $\wedge$ $\mathbf{B}_p$ Fresh(msg) $\rightarrow$ $\mathbf{B}_p$ Reliable(msg).
P5. Receive$(p, (msg_1, msg_2))$ $\leftrightarrow$ (Receive(p, msg_1) $\wedge$ Receive(p, msg_2)).

Here $\{msg\}k$ means that message *msg* is encrypted with key k. The meanings of these axioms are as follows: P0 says that, if principal p believes the key k is shared with another principal q and receives message *msg* encrypted with k, then p believes that q sent *msg*. P1 says that, if principal p believes the public key k belongs to another principal q and receives message *msg* encrypted with k, then p believes that q sent *msg*. P2 says that, if principal p believes his own secret key k and receives message *msg* encrypted with k, then p believes the message is reliable. P3 says that if principal p believes another principal q once sent message *msg*, and p believes that message *msg* s fresh, then p believes that q believes the message *msg* is reliable. P4 says that if principal p believes that another principal q sent message *msg*, and p believes that *msg* is fresh, then p believes the message *msg* is reliable. The meaning of P5 is straightforward.

Based on these postulates we can reason about evolving agent beliefs involved in an authentication system. Assuming $\mathbf{B}_p$ Shared(p, q, k) and **first next** Receive$(p, \{msg\}k)$ hold, i.e., we assume that principle p believes key k is shared with another principle q and, at the next moment after the initial time, p has received a message *msg* encrypted by key k. Then we can show that p will believe that the message was sent by q. The proof outline is given as follows:

(1) $\mathbf{B}_p$ Shared(p, q, k) (assumption)
(2) **next** $\mathbf{B}_p$ Shared(p, q, k) (from (1), by rule IR4)
(3) **first next** $\mathbf{B}_p$ Shared(p, q, k) (from (2), by rule IR4)
(4) **first next** Receive$(p, \{msg\}k)$ (assumption)
(5) **first next** $\mathbf{B}_p$ Shared(p, q, k) $\wedge$ **first next** Receive$(p, \{msg\}k)$
(from (3) & (4), by IR6)
(6) **first next** ($\mathbf{B}_p$ Shared$(p, q, k) \wedge$ Receive$(p, \{msg\}k)$) (from (5), by axiom A2)
(7) **first next** ($\mathbf{B}_p$ Shared(p, q, k) $\wedge$ Receive$(p, \{msg\}k)$ $\rightarrow$ $\mathbf{B}_p$ Send(q, msg))
(from A0, by repeated rule IR4)
(8) **first next** ($\mathbf{B}_p$ Shared(p, q, k) $\wedge$ Receive$(p, \{msg\}k)$) $\rightarrow$
first next ($\mathbf{B}_p$ Send(q, msg)) (from (7), by axiom A1)
(9) **first next** ($\mathbf{B}_p$ Send(q, msg)) (from (6) & (8), by rule IR1)
□

The last formula is what we want to show. This proof process involves the use of axioms and inference rules of TML$^+$, including those rules on temporal operators as well as modus ponens (IR1).

3 Theory of trust for Kerberos

A theory of trust provides a foundation for reasoning about agent beliefs. In this section, we show how to establish a theory of trust for a given system based on the logic TML$^+$. In particular, we propose a method to build a theory of trust for the well-known Kerberos protocol, and discuss and reason about the security properties of the theory.

3.1 Kerberos authentication system

In an open network computing environment, a workstation cannot be trusted to identify its users correctly to network services. Kerberos provides an alternative approach whereby a trusted third-party authentication service is used to verify users' identities (Steiner et al. 1988). The Kerberos protocol establishes a shared key between two principals with help from an authentication server (Miller et al. 1987). It is based on the shared-key Needham–Schroeder protocol (Needham and Schroeder 1978), but it makes use of timestamps to improve security (Denning and Sacco 1981).

For analysis of the Kerberos system, we use the notations C, AS, TGS and Serv to denote client, Authentication Server, Ticket Granting Server and server respectively; TGT and SGT denote specific tickets; k_c, k_{serv}, k_{tgs}, $k_{c,tgs}$, $k_{c,serv}$ denote specific keys; $Timestamp_c$ denotes a timestamp sent by the client; and A_c denotes an authenticator sent by the client.

The Kerberos authentication process is as follows (see Fig. 2):

1. *Requesting a Ticket Granting Ticket (TGT)*: The client sends a request to the Authentication Server (AS). The request contains the client's name and the name of the ticket-granting server (TGS).
2. *Granting a TGT*: The AS checks whether the client is in its database. If the client is found, the AS generates a random session key to be used between the client and the TGS. The session key has a lifetime of a few hours. And it also creates a TGT

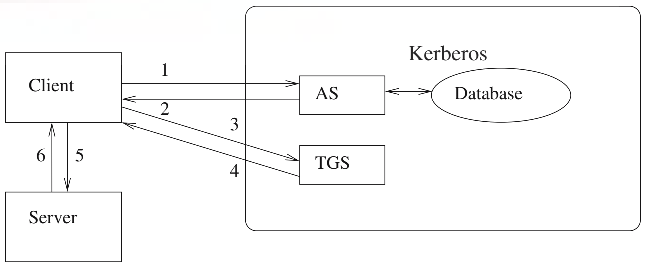


Fig. 2 Kerberos Authentication system

for the TGS which is encrypted by the key k_{tgs}. The AS then sends the ticket and the session key back to the client. This response is encrypted by the key k_c.

3. *Requesting a Service Granting Ticket* (*SGT*): The Client sends a request to the TGS in order to access a server. The request contains the name of the server, along with the TGT and an authenticator.
4. *Granting a SGT*: The ticket-granting server then checks the authenticator and TGT. If valid, the TGS generates a random session key to be used between the client and the server. The session key has a lifetime of a few minutes. And it also creates a SGT for the server which is encrypted by the key k_{serv}. The TGS then sends the ticket and the session key back to the client. This response is encrypted by the key $k_{c,tgs}$.
5. *Requesting a service*: The Client sends a request to the server in order to obtain a service. The request contains the TGT and an authenticator.
6. *Granting a service*: The server checks the authenticator and SGT. If valid, it allows the request to proceed. And finally, if the client specifies that it wants the server to prove its identity too, the server adds one to the timestamp the client sent within the authenticator, and sends it back to the client. This response is encrypted by the key $k_{c,serv}$.

3.2 Establishing a theory of trust for Kerberos

In the Kerberos system, in order to obtain a service, a client must first send a request to the AS for obtaining a Ticket Granting ticket, then send a request to the TGS for obtaining a Service Granting Ticket, and finally send a request to the server for obtaining a service.

As shown in Fig. 3, we consider four different states within the authentication process: s_0 (initial), s_1(TGT granted), s_2 (SGT granted), s_3 (service granted). During the authentication process, the client must be at one of the four states. The methods of authentication adopted for the AS, TGS, and server are assumed to be m_1, m_2 and m_3, respectively. Each agent accepts a request from a client only if it believes that the client is properly authenticated by the corresponding authentication method.

This is a multi-agent authentication system. The system uses such security mechanisms to keep the communication secure. This implies that principals would trust the security mechanisms of this system. Initially, agents should have trust in:

- The capability of authentication agents to work as required. In Kerberos, these agents are AS, TGS, Serv.
- Security mechanisms used within the system, which may include encryption functions, the freshness-check mechanisms for timestamps, etc.
- The security of encryption keys.
- The freshness of timestamps. This implies the principal's clocks are synchronized with the server's clock.

That is, the principals involved in the system must have an initial trust in the set of security mechanisms, denoted by $\mathscr{M}$,

$$\mathscr{M} = \{\text{AS, TGS, Serv}, \textit{TimeStamps}, \\ k_c, k_{serv}, k_{tgs}, k_{c,tgs}, k_{c,serv}\}.$$

Here we consider two kinds of clients, registered clients and unregistered clients; any unregistered client will be regarded as an intruder by the Kerberos authentication system.

In order to establish a theory of trust for the system, we further define the following predicates.

- $\text{AtState}(c, s)$: Client c is at an authentication state s ranging over $\{s_0, s_1, s_2, s_3\}$.
- $\text{Request}(c, s)$: Client c requests to enter state s.
- $\text{Authenticated}(c, m)$: The request of client c is authenticated by authentication method m, where m represents an authentication method ranging over $\{m_1, m_2, m_3\}$. m_1, m_2 and m_3 are defined by rules R8, R9 and R10 respectively (see below).
- $\text{IsAclient}(c)$: c is a client of the system.
- $\text{TicketGranted}(c, t)$: Client c is granted a ticket t ranging over {TGT, SGT}.
- $\text{ServiceGranted}(c, sv)$: Client c is granted a service sv.
- $\text{RequireMutualAuth}(c)$: Mutual authentication is required by client c.
- $\text{Expired}(k)$: key k is expired.
- $\text{Compromised}(k)$: Key k is compromised.
- $\text{IsSecured}(m)$: Authentication method m is secured.

First we have the following rule that describes the functional (behavioural) properties of the authentication system.R0.

R0. $\text{AtState}(c, s_{i-1}) \wedge \text{Request}(c, s_i) \rightarrow$

$$(\textbf{next}\text{AtState}(c, s_i) \leftrightarrow \mathbf{B}_{a_i}\text{Authenticated}(c, m_i)), \quad \text{for} \quad i = 1, 2, 3.$$

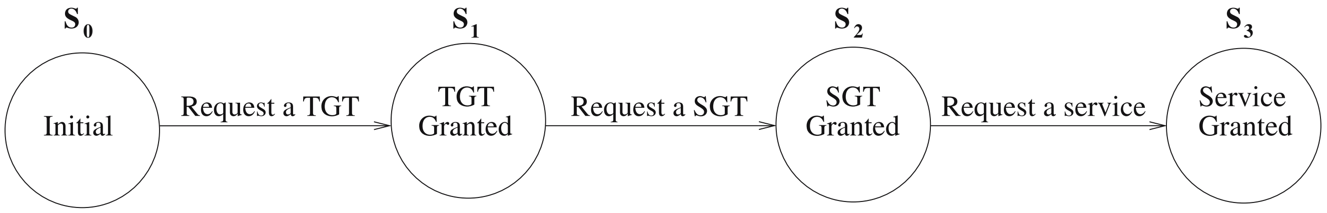


Fig. 3 Four states within authentication process

The rule R0 specifies the authentication procedure. It says that if currently the client c is at authentication state s_{i-1} and requests to enter the authentication state s_i, then the client is at s_i at the next moment in time if and only if agent a_i believes the client is authenticated by authentication method m_i. Here a_i represents an authentication agent for agent c ranging over $\{a_1, a_2, a_3\}$ where a_1, a_2 and a_3 represent Authentication Server, Ticket Granting Server and server respectively; m_i represents an authentication method ranging over $\{m_1, m_2, m_3\}$. We extend the set of security mechanisms by adding all authentication methods, now $\mathscr{M} = \{\text{AS, TGS, Serv}, m_1, m_2, m_3, TimeStamps, k_c, k_{serv}, k_{tgs}, k_{c,tgs}, k_{c,serv}\}$.

Rules R1–R4 specify the authentication states 0, 1, 2 and 3 respectively.

R1. $\text{AtState}(c, s_0) \leftrightarrow \neg\,\text{TicketGranted}(c, \text{ TGT})$.
R2. $\text{AtState}(c, s_1) \leftrightarrow \text{TicketGranted}(c, \text{ TGT})$.
R3. $\text{AtState}(c, s_2) \leftrightarrow \text{TicketGranted}(c, \text{ SGT})$.
R4. $\text{AtState}(c, s_3) \leftrightarrow \text{ServiceGranted}(c, sv)$.

Rules R5–R7 specify clients' requests for entering states 1, 2 and 3 respectively, for example, R7 says that for requesting a service a client sends a SGT and an authenticator to a server, we assume that the server receives the request simultaneously.

R5. $\text{Request}(c, s_1) \leftrightarrow \text{Send } (c, \{c, \text{TGS}\}) \wedge \text{Receive (AS}, \{c, \text{TGS}\})$.
R6. $\text{Request}(c, s_2) \leftrightarrow \text{Send } (c, (\text{Serv}, \{\text{TGT}\}k_{tgs}, \{\text{A}_c\}k_{c,tgs})) \wedge \text{Receive (TGS}, (\text{Serv}, \{\text{TGT}\}k_{tgs}, \{\text{A}_c\}k_{c,tgs}))$.
R7. $\text{Request}(c, s_3) \leftrightarrow \text{Send } (c, (\{\text{SGT}\}k_{serv}, \{\text{A}_c\}k_{c,serv})) \wedge \text{Receive (Serv}, (\{\text{SGT}\}k_{serv}, \{\text{A}_c\}k_{c,serv}))$.

Rules R8–R10 specify the authentication methods m_1, m_2 and m_3 adopted by AS, TGS and Server respectively.

R8. $\mathbf{B}_{\text{AS}}\,\text{Authenticated}(c, m_1) \leftrightarrow \text{Receive (AS}, \{c, \text{TGS}\}) \wedge \mathbf{B}_{\text{AS}}\,\text{IsAclient}(c)$.
R9. $\mathbf{B}_{\text{TGS}}\,\text{Authenticated}(c, m_2) \leftrightarrow \mathbf{B}_{\text{TGS}}\,(\text{Reliable}(A_c) \wedge \text{Reliable(TGT)})$.
R10. $\mathbf{B}_{\text{Serv}}\,\text{Authenticated}(c, m_3) \leftrightarrow \mathbf{B}_{\text{Serv}}\,(\text{Reliable}(A_c) \wedge \text{Reliable(SGT)})$.

Rules R11–R13 specify the granting processes of a TGT, a SGT and a service respectively.

R11. $\mathbf{B}_{\text{AS}}\,\text{Authenticated}(c, m_1) \rightarrow \textbf{next}\,\text{Receive } (c, \{k_{c,tgs}, \{\text{TGT}\}k_{tgs}\}k_c)$.
R12. $\mathbf{B}_{\text{TGS}}\,\text{Authenticated}(c, m_2) \rightarrow \textbf{next}\,\text{Receive } (c, \{k_{c,serv}, \{\text{SGT}\}k_{serv}\}k_{c,tgs})$.
R13. $\mathbf{B}_{\text{Serv}}\,\text{Authenticated}(c, m_3) \rightarrow \textbf{next}\,(\text{ServiceGranted}(c, sv) \wedge (\text{RequireMutualAuth}(c) \rightarrow \text{Receive } (c, \{TimeStamp_c + 1\}k_{c,serv})))$.

Now we have established a theory T = {R0, R1, R12, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, R13} which is applied for this authentication system. Since agents have trust in the security mechanisms of Kerberos, the trust theory can therefore be used for reasoning about agent beliefs about the system.

The above example has shown a whole procedure of establishing a theory of trust for a given authentication system. The procedure involves the following steps:

1. Analysing the authentication process within the given authentication system, identify each authentication state.
2. Analysing the system to identify the trusted agents in it, and finding what kind of a trust and trust relationships involved in the system.
3. Defining appropriate predicates used to express agent beliefs.
4. Defining rules that describe functions/behaviours of the trusted agents.

Correctly identifying trusted agents is the basis for constructing a valid theory for the system. Defining appropriate predicates would be helpful to simplify the procedure of constructing the theory, so that the theory would be easily understood. Finally, we need to check the consistency of those rules in the theory for obtaining a valid theory for the system.

The soundness of a theory of trust is an important issue for establishing a valid theory. In order to guarantee a theory to be sound, we need to check every rule to make sure that it is consistent with those rules that have been in the theory. Completeness for a theory of trust is also an important issue. It is involved in the analysis of security requirements and fulfilling these requirements.

3.3 Analysing security properties for Kerberos

Now we discuss the security properties of the theory, which focus on the confidentiality of those keys used to encrypt tickets (SGT and TGT) and authenticators. We assume that:

- Tickets: If an intruder does not know those secret keys (k_c, k_{tgs} and k_s) used to encrypt a ticket, in other words, secret keys are not compromised, then the intruder cannot learn those tickets.
- Authenticators: If an intruder does not know those shared keys ($k_{c,tgs}$ and $k_{c,srev}$) used to encrypt an authenticator, then the intruder cannot learn those authenticators.

We also give the following assumptions:

$\mathbf{B}_{\mathrm{C}}$ Owner(c, k)	$\mathbf{B}_{\mathrm{C}}$ Owner(tgs, k)
$\mathbf{B}_{\mathrm{C}}$ Owner($serv$, k)	$\mathbf{B}_{\mathrm{AS}}$ Owner(c, k)
$\mathbf{B}_{\mathrm{AS}}$ Owner(tgs, k)	$\mathbf{B}_{\mathrm{TGS}}$ Owner(tgs, k)
$\mathbf{B}_{\mathrm{TGS}}$ Owner($serv$, k)	$\mathbf{B}_{\mathrm{Serv}}$ Owner($serv$, k)
$\mathbf{B}_{\mathrm{C}}$ Shared(c, tgs, k)	$\mathbf{B}_{\mathrm{C}}$ Shared(c, $serv$, k)
$\mathbf{B}_{\mathrm{AS}}$ Shared(c, tgs, k)	$\mathbf{B}_{\mathrm{TGS}}$ Shared(c, tgs, k)
$\mathbf{B}_{\mathrm{TGS}}$ Shared(c, $serv$, k)	$\mathbf{B}_{\mathrm{Serv}}$ Shared(c, $serv$, k)

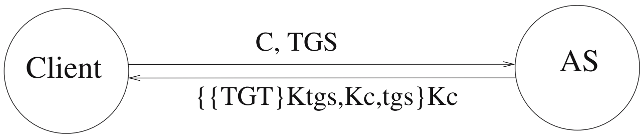


Fig. 4 Granting a TGT

The first eight assumptions are about secret keys used by clients, AS, TGS and the server. The last six assumptions are about session keys. And we also assume that:

$\mathbf{B}_{\mathrm{C}}$ Fresh($Timestamp_{as}$)	$\mathbf{B}_{\mathrm{C}}$ Fresh($TimeStamp_{tgs}$)
$\mathbf{B}_{\mathrm{C}}$ Fresh($TimeStamp_{serv}$)	$\mathbf{B}_{\mathrm{TGS}}$ Fresh($TimeStamp_c$)
$\mathbf{B}_{\mathrm{TGS}}$ Fresh($TimeStamp_{as}$)	$\mathbf{B}_{\mathrm{Serv}}$ Fresh($TimeStamp_c$)
$\mathbf{B}_{\mathrm{Serv}}$ Fresh($TimeStamp_{tgs}$)	

These assumptions show that the client, TGS, and the server believe that timestamps generated elsewhere are fresh. It indicates that the authentication system relies on the use of synchronized clocks.

In analysing the authentication processes, we give two examples as follows.

Example—Granting a TGT Assume that AtState(c, s_0) $\wedge$ Request(c, s_1), and we also assume IsAclient(c) (as shown in Fig. 4), then we can prove **next** AtState(c, s_1).
The proof outline is given as follows:

(1)	AtState(c, s_0) $\wedge$ Request(c, s_1)	(assumption)
(2)	Request(c, s_1)	(from (1), by IR7)
(3)	Send (c, (c, TGS)) $\wedge$ Receive (AS, (c, TGS))	(from (2) & R5, by IR8)
(4)	Receive (AS, (c, TGS))	(from (3), by IR7)
(5)	IsAclient(c)	(assumption)
(6)	$\mathbf{B}_{\mathrm{AS}}$ IsAclient(c)	(from (5),by IR5)
(7)	Receive (AS, (c, TGS)) $\wedge$ $\mathbf{B}_{\mathrm{AS}}$ IsAclient(c)	(from (5) & (6), by IR6)
(8)	$\mathbf{B}_{\mathrm{AS}}$ Authenticated(c, m_1)	(from (7) & R8 by IR8)
(9)	**next** AtState(c, s_1) $\leftrightarrow$ $\mathbf{B}_{\mathrm{AS}}$ Authenticated(c, m_1)	(from (1) & R0, by IR1)
(10)	**next** AtState(c, s_1)	(from (8) & (9), by IR8)

□

We have proved **next** AtState(c, s_1). This concludes the analysis of the TGT granting process.

Example—Granting a SGT Assume that AtState(c, s_1) $\wedge$ Request(c, s_2) (as shown in Fig. 5), then we can prove **next** AtState(c, s_2).

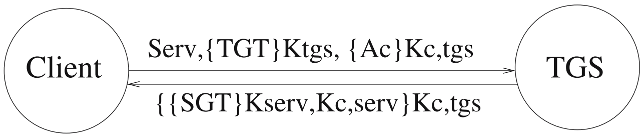


Fig. 5 Granting a SGT

The proof outline is given as follows:

(1) $\text{AtState}(c, \text{s}_1) \wedge \text{Request}(c, \text{s}_2)$ (assumption)
(2) $\text{Request}(c, \text{s}_2)$ (from (1), by IR7)
(3) $\text{Send}(c, (\text{Serv}, \{\text{TGT}\}k_{tgs}, \{\text{A}_c\}k_{c,tgs})) \wedge$
$\text{Receive}(\text{TGS}, (\text{Serv}, \{\text{TGT}\}k_{tgs}, \{\text{A}_c\}k_{c,tgs}))$ (from (2) & R6, by IR8)
(4) $\text{Receive}(\text{TGS}, (\{\text{TGT}\}k_{tgs}, \{\text{A}_c\}k_{c,tgs}))$ (from (3), by IR7)
(5) $\text{Receive}(\text{TGS}, \{\text{TGT}\}k_{tgs})$ (from (4), by P5)
(6) $\mathbf{B}_{\text{TGS}}\ \text{Owner}(\text{TGS}, k_{tgs})$ (assumption)
(7) $\mathbf{B}_{\text{TGS}}\ \text{Owner}(\text{TGS}, k_{tgs}) \wedge \text{Receive}(\text{TGS}, \{\text{TGT}\}k_{tgs})$ (from (5) & (6), by IR6)
(8) $\mathbf{B}_{\text{TGS}}\ \text{Reliable}(\text{TGT})$ (from (7) & P2, by IR1)
(9) $\text{Receive}(\text{TGS}, \{\text{A}_c\}k_{c,tgs})$ (from (4), by P5)
(10) $\mathbf{B}_{\text{TGS}}\ \text{Shared}(c, \text{TGS}, k_{c,tgs})$ (assumption)
(11) $\mathbf{B}_{\text{TGS}}\ \text{Shared}(c, \text{TGS}, k_{c,tgs}) \wedge \text{Receive}(\text{TGS}, \{\text{A}_c\}k_{c,tgs})$ (from (9) & (10), by IR6)
(12) $\mathbf{B}_{\text{TGS}}\ \text{Send}(c, \text{A}_c)$ (from (11) & P0, by IR1)
(13) $\mathbf{B}_{\text{TGS}}\ \text{Fresh}(\mathit{TimeStamp}_c)$ (assumption)
(14) $\mathbf{B}_{\text{TGS}}\ \text{Send}(c, \text{A}_c) \wedge \mathbf{B}_{\text{Serv}}\ \text{Fresh}(\mathit{TimeStamp}_c)$ (from (12) & (13), by IR6)
(15) $\mathbf{B}_{\text{TGS}}\ \text{Reliable}(\text{A}_c)$ (from (14) & P4, by IR1)
(16) $\mathbf{B}_{\text{TGS}}\ \text{Reliable}(\text{TGT}) \wedge \mathbf{B}_{\text{Serv}}\ \text{Reliable}(\text{A}_c)$
(from (8) & (15), by IR6)
(17) $\mathbf{B}_{\text{TGS}}\ \text{Authenticated}(c, m_2)$ (from (16) & R10, by IR8)
(18) $\mathbf{next}\ \text{AtState}(c, \text{s}_1) \leftrightarrow \mathbf{B}_{\text{AS}}\ \text{Authenticated}(c, m_1)$ (from (17) and R0, by IR1)
(19) $\mathbf{next}\ \text{AtState}(c, \text{s}_2)$ (from (17) & (18), by IR8)

□

We have proved **next** AtState(c, s$_2$). This concludes the analysis of the SGT granting process.

The analysis of the service granting process can be achieved in the same way. The above examples demonstrate the security properties of Kerberos. The system is secure as long as those keys are secure.

Assume that a client c completed the Authentication Service Exchange with AS and obtained a TGT from it. Then he uses this ticket to make a request for a SGT. TGS receives this request and grants a SGT which is encrypted by the key k_{serv}, then the client sends a request to the server which includes the SGT and his authenticator encrypted by the key $k_{c,serv}$. If an intruder *eve* later masquerades the client c, she may duplicate the request and forward it to the server, but she is unable to create a new authenticator if she does not know the key $k_{c,serv}$. And the server will not accept her request because of the repeated authenticator. This prompts an error message from the server which is sent to the client. As a result, the client will be notified that an intruder tried to masquerade his identity.

The security property of the Kerberos authentication system can be expressed by the following propositions.

Proposition 1 (*Confidentiality of Authentication State One*)

$$\mathbf{B}_{\text{AS}}\ \text{IsSecured}(m_1) \leftrightarrow (\text{IsAclient}(c) \wedge \neg\ \text{Compromised}(k_c)).$$

This proposition assures that the AS believes the authentication method m_1 is secured iff c is a registered client and the key k_c is not compromised.

Proposition 2 (*Confidentiality of Authentication State Two*)

$$\mathbf{B}_{\text{TGS}}\ \text{IsSecured}(m_2) \leftrightarrow \neg(\text{Expired}(k_{c,tgs}) \vee \text{Compromised}(k_{tgs})).$$

This proposition assures that the TGS believes the authentication method m_2 is secured iff the key $k_{c,tgs}$ is not expired and the key k_{tgs} is not compromised.

Proposition 3 (*Confidentiality of Authentication State Three*)

$\mathbf{B}_{\mathrm{Serv}}\ \mathrm{IsSecured}(m_3) \leftrightarrow \neg(\mathrm{Expired}(k_{c,serv}) \vee \mathrm{Compromised}(k_{serv}))$.

This proposition assures that the server believes the authentication method m_3 is secured iff the key $k_{c,serv}$ is not expired and the key k_{serv} is not compromised.

4 Trust theory revision

The theory of trust for a given system is built based on the initial trust of agents in the security mechanisms of the system or the initial trust state. However, in a dynamic environment, an agent may lose its trust or gain new trust to some degree at any time. For example, for the last two days John trusts Bob, but this morning John found that Bob lied to him, so John no longer trusts Bob. Once there are some changes to the trust of agents in the security mechanisms of the system, the theory established based on the initial trust state will no longer be valid. In this case, the theory must be revised or not be used for any security purpose (Liu et al. 2004; Ma and Orgun 2006b, 2007). Therefore, in considering theory revision, we first need a method for modelling the dynamics of trust and a technique for expressing trust changes.

For modelling trust changes, let μ be a multi-agent system, Ω be the set of agents involved in a system (e.g., users, operators, managers *etc*) and Θ be the set of binary trust relations over Ω. That is, Θ consists of pairs of agents such that $(x, y) \in \Theta$ if and only if *x trusts y*. We give the formal definition of a trust state as follows: A *trust state* of a system, denoted by TS, is defined as the pair $\mathrm{TS} = (\Omega, \Theta)$. Then, we define a predicate $\mathrm{Trust\_State}(\mathrm{TS}, \mu)$ meaning that TS is the trust state of the system μ. Therefore if we have the formula **first** $\mathrm{Trust\_State}(\mathrm{TS}_0, \mu)$, TS_0 is the initial trust state of μ.

Jonker and Treur (1999) call an event that can influence the degree of trust of an agent a *trust-positive experience* or a *trust-negative experience* of the agent. If it is a trust-positive experience then the agent may gain his trust to some degree; if the event an agent experiences is a trust-negative experience then the agent may lose his trust to some degree. In this paper, we revise and analyse theories of trust based on the binary valued model which only has two different trust degrees, 1 (*trust*) and 0 (*no trust*). Trust changes to a trust state TS involves two cases: agents lose their trust in some agents (security mechanisms) in the current state TS, and agents gain new trust in some agents (e.g., some new security mechanisms are adopted). Following the method in (Ma and Orgun 2006b), we view a trust change to trust state TS as consisting of two classes of operations: deleting a relation (*x*,*y*) from TS, and adding a relation (*x*,*y*) to TS. Therefore, we express a trust change to state S as a pair of sets: one set contains all trust relations of the form (*x*,*y*) that will be added to state TS; and the other set contains all trust relations of the form (*x*,*y*) that will be deleted from state TS. Formally, we say that: $\delta = (\mathrm{IN}, \mathrm{OUT})$ is a *trust change* to a trust state $\mathrm{TS} = (\Omega, \Theta)$, if IN and OUT satisfy the following conditions: (1) $\mathrm{OUT} \subseteq \Theta$, and (2) $\mathrm{IN} \cap \Theta = \emptyset$. IN and OUT are called the *in set* and *out set*, respectively.

In a dynamic environment, an agent may lose its trust or gain new trust at any moment in time after it has gained the initial trust. Once agents lose their trust in the security mechanisms of a system, then the theory based on the initial trust of the system is no longer valid, therefore we have the following principles:

- Given a system μ, suppose that we have **first next**$^{(i)}$ $\mathrm{Trust\_State}(\mathrm{TS}_i, \mu)$, for $i = 0, 1, 2, \ldots$, and T_0 is a sound theory of trust for μ that specifies the initial trust of agents in the security mechanisms, then, if from time 0 until some time t, $\mathrm{TS}_i = \mathrm{TS}_0$ for any $0 \le i \le t$, then the theory T_0 is valid for the whole time interval $[0, t]$ and, therefore, any statement derived from the theory would be regarded as valid related to the period $[0, t]$.
- Given a system μ, suppose that we have **first next**$^{(i)}$ $\mathrm{Trust\_State}(\mathrm{TS}_i, \mu)$, for $i = 0, 1, 2, \ldots$, and T_0 is a sound theory of trust for μ that specifies the initial trust of agents in the security mechanisms TS_0. If at some time, say t, TS_0 does not hold, i.e., there is some agent losing its trust in the mechanisms, then the theory must be revised.

Assume that we have $\mathrm{Trust\_State}(\mathrm{TS}, \mu) \wedge$ **next** $\mathrm{Trust\_State}(\mathrm{TS}', \mu)$, then there are two possible cases to consider:

- $\mathrm{TS} = \mathrm{TS}' \leftrightarrow (\mathrm{IN} = \emptyset \wedge \mathrm{OUT} = \emptyset)$;
- $\mathrm{TS} \neq \mathrm{TS}' \leftrightarrow (\mathrm{IN} \neq \emptyset \vee \mathrm{OUT} \neq \emptyset)$.

In the case when $\mathrm{TS} = \mathrm{TS}'$, the trust state at the next moment is the same as that at the current moment, so there are no changes to the trust of the system; while for the case when $\mathrm{TS} \neq \mathrm{TS}'$, there must be some changes to the trust of the system. We assume that, if an agent, say a, has any *trust-negative experience* with agent b in whom a trusted before, then at that moment a loses trust in b and the tuple (a, b) will therefore be deleted from the system. Similarly, when agent a gains trust in a new agent b due to any *trust-positive experience*, the tuple (a, b) will be added.

The change with adding/deleting trust relations can be expressed as follows:

$$(\text{Trust\_State}(\text{TS}, \mu) \wedge (\text{IN} \neq \emptyset \vee \text{OUT} \neq \emptyset)) \rightarrow \textbf{next}\ \text{Trust\_State}(\text{TS}', \mu)$$

where $\text{TS} = (\Omega, \Theta)$ and $\text{TS}' = (\Omega \cup \{y | (x, y) \in \text{IN} \wedge x \in \Omega \wedge y \notin \Omega\}, (\Theta \setminus \text{OUT}) \cup \text{IN})$.

Changes to the trust state require theory revision. The following example shows how to obtain an evolving theory of trust based on the analysis of the dynamics of trust.

Recall the example theory given in Section 4, T = {R0, R1, R12, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, R13}. Let $\text{T}_0 =$ T and the initial trust state be TS_0. Then T_0 is the theory that specifies trust of agents in the set of security mechanisms $\mathscr{M} = \{\text{AS, TGS, Serv}, m_1, m_2, m_3, \mathit{TimeStamps}, k_c, k_{serv}, k_{tgs}, k_{c,tgs}, k_{c,serv}\}$ at the trust state TS_0.

Assume that it is found that the authentication method m_1 is not secure enough. We improve m_1 by requiring that a client must encrypt its request. Since R5 was authentication method m_1, it must be revised. Also, because of the modification of R5, we have to modify R8. Furthermore, we change R13 to make mutual authentication compulsory. We then rewrite rules R5, R8, R13 as follows:

R5' $\text{Request}(c, s_1) \leftrightarrow \text{Send}\ (c, \{c, \text{TGS}\}k_c) \wedge \text{Receive}\ (\text{AS}, \{c, \text{TGS}\}k_c)$.

R8' $\mathbf{B}_{\text{AS}}\ \text{Authenticated}(c, m_1) \leftrightarrow \text{Reliable}\ (\{c, \text{TGS}\}) \wedge \mathbf{B}_{\text{AS}}\ \text{IsAclient}(c)$.

R13' $\mathbf{B}_{\text{Serv}}\ \text{Authenticated}(c, m_3) \rightarrow \textbf{next}(\text{Service Granted}(c, sv) \wedge \text{Receive}\ (c, \{\mathit{TimeStamp}_c + 1\}k_{c,serv}))$.

Then, we obtain a revised theory T' = {R0, R1, R12, R3, R4, R5', R6, R7, R8', R9, R10, R11, R12, R13'}. Those revisions improve the security of the system. Since systems change and evolve, there is a need to monitor trust relationships, to determine whether those rules still apply and whether the security mechanisms can still satisfy security requirements, in order to make decisions on theory revision.

5 Conclusions and future work

We have proposed a formal approach to specifying agent beliefs, constructing and revising evolving theories of trust for authentication protocols. Our approach includes techniques for modelling trust state and trust changes within authentication systems, and it could be useful in designing, implementing and verifying authentication protocols.

It has been argued that any logical system modelling active agents should be a combined system of logics of knowledge, belief, time and norms (Liu et al. 2004). Combining logics is therefore emerging as an active research area in formal methods. In order to analyse authentication protocols it is necessary to have a logic that can satisfactorily deal with all dynamic aspects of authentication systems. As future work, we plan to investigate different combined logics of belief and time for constructing trust theories for authentication protocols. Fibring (Gabbay 2000) could also be very useful for our purpose to combine logics because it gives all the logics equal status in the combination. Preliminary work on fibring SLTL and TML is reported by (Orgun et al. 2006).

Future work would also involve the following aspects:

- Investigating methods for studying security properties of dynamic systems, together with the study of related reasoning techniques based on different trust models.
- Implementation of an automated process of theory revision; such an implementation may include an automatic or a semi-automatic collection of information about trust changes and automatic production of theory revision.
- Investigation of the soundness and completeness of a theory of trust.

Acknowledgements This research was supported in part under Australian Research Council's Discovery Projects funding scheme (project numbers DP0452628 and DP0559592) and a Macquarie University Research Development Grant (MQRDG).

References

Burrows, M., Abadi, M., & Needham, R. M. (1990). A logic of authentication. *ACM Transactions on Computer Systems*, *8*(1), 18–36.

Campbell, E. A., Safavi-Naini, R., & Pleasants, P. A. (1992). Partial belief and probabilistic reasoning in the analysis of secure protocols. In *Proceedings of the 5th IEEE computer security foundations workshop* (pp. 84–91). IEEE Computer Society Press.

Denning, D. E., & Sacco, G. M. (1981). Timestamps in key distribution protocols. In *Commun. ACM*, vol. 24.8 (pp. 533–536).

Finger, M., & Gabbay, D. M. (1992). Adding a temporal dimension to a logic system. *Journal of Logic, Language and Information*, *1*, 203–233.

Fisher, M., & Ghidini, C. (2002). Agents with bounded temporal resources. In Foundations and applications of multi-agent systems. UKMAS workshops 1996-2000. Selected papers, LNAI, vol. 2403 (pp. 169–184). Springer.

Fitting, M., & Mendelsohn, R. L. (1999). *First-order modal logic*. Kluwer Academic Publishers.

Gabbay, D. M. (2000). Fibring logics. *Journal of Logic, Language and Information*, *9*(4), 511–513.

Gabbay, D., Pigozzi, G., & Woods, J. (2003). Controlled revision—an algorithmic approach for belief revision. *Journal of Logic and Computation*, *13*(1), 3–22.

Giacomo, G. D., Lenzerini, M., Poggi, A., & Rosati, R. (2006). On the update of description logic ontologies at the instance level. In *Proceedings of the twenty-first national conference on artificial intelligence*. AAAI Press.

Gong, L., Needham, R., & Yahalom, R. (1990). Reasoning about belief in cryptographic protocols. In *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, (pp. 234–248). IEEE Computer Society Press.

Hughes, G. E., & Cresswell, M. J. (1996). *A New Introduction to Modal Logic*. London: Routledge.

Jonker, C. M., & Treur, J. (1999). Formal analysis of models for the dynamics of trust based on experiences. In *Proceedings of Multi-Agent System Engineering '99*, vol. 1647 (pp. 221–231). Springer.

Kripke, S. A. (1963) Semantical considerations on modal logic. *Acta Philosophica Fennica*, *16*, 83–94.

Liu, C. (2001). Logical foundations for reasoning about trust in secure digital communication. In *Proceedings of AI2001: Advances in Artificial Intelligence. LNCS*, vol. 2256 (pp. 333–344). Springer.

Liu, C., & Orgun, M. A. (1996). Dealing with multiple granularity of time in temporal logic programming. *Journal of Symbolic Computation*, *22*(5/6), 699–720.

Liu, C., & Orgun, M. A. (1999). Verification of reactive systems using temporal logic with clocks. *Theoretical Computer Science*, *220*(2), 377–408.

Liu, C., & Ozols, M. A. (2002). Trust in secure communication systems—the concept, representations, and reasoning techniques. In *Proceedings of AI2002: Advances in artificial intelligence. LNCS*, vol. 2557 (pp. 60–70). Springer.

Liu, C., Ozols, M. A., & Orgun, M. A. (2004). A temporalised belief logic for specifying the dynamics of trust for multi-agent systems. In *Proceedings of the Ninth Asian Computer Science Conference 2004. LNCS*, vol. 3321 (pp. 142–156). Springer.

Liu, H., Lutz, C., Milicic, M., & Wolter, F. (2006). Updating description logic aboxes. In *Proceedings of International Conference of Principles of knowledge Representation and Reasoning (KR)* (pp. 46–56).

Ma, J., & Orgun, M. (2006a). Theories of trust for authentication systems. In *2nd Secure Knowledge Management Workshop* (unpaginated cd-rom proceedings).

Ma, J., & Orgun, M. (2006b). Trust management and trust theory revision. *IEEE Transactions on Systems, Man and Cybernetics, Part A*, *36*(3), 451–460.

Ma, J., & Orgun, M. (2007). Specifying agent beliefs for authentication systems. In *Proceedings of 4th European Conference on Universal Multiservice Networks*, (pp. 410–418). IEEE Computer Society Press.

McCarthy, J. (1990). *Formalizing common sense: Papers by John McCarthy (V. Lifschitz)*. Ablex Publishing Corporation.

Meyer, T. (1999). Basic infobase change. In *Proceedings of AI'99. LNCS*, vol. 1747 (pp. 156–167). Springer.

Miller, S. P., Neuman, C., Schiller, J. I., & Saltzer, J. H. (1987). Kerberos authentication and authorization system. *Project Athena Technical Plan, Sect. E.2.1. MIT*.

Moser, L. (1989). A logic of knowledge and belief for reasoning about computer security. In *Proceedings of the Computer Security Foundations Workshop II* (pp. 57–63). IEEE Computer Society Press.

Needham, R. M., & Schroeder, M. D. (1978). Using encryption for authentication in large networks of computers. In *Commun. ACM*, vol. 21 (pp. 993–999).

Oorschot, P. C. V. (1993). Extending cryptographic logics of belief to key agreement protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, (pp. 233–243). ACM Press.

Orgun, M. A., Ma, J., Liu, C., & Governatori, G. (2006). Analysing stream authentication protocols in autonomous agent-based systems. In *Proceedings of the 2nd International Symposium on Dependable Autonomic and Secure Computing* (pp. 325–332). IEEE Computer Society Press.

Rangan, P. V. (1988). An axiomatic basis of trust in distributed systems. In *Proceedings of the 1988 IEEE computer Society Symposium on Research in Security and Privacy*, (pp. 204–211).

Schulte, O. (1999). Minimal belief change and pareto-optimality. In *Proceedings of AI'99. LNCS*, vol. 1747 (pp. 144–155). Springer.

Steiner, J. G., Neuman, B. C., Schiller, J. (1988). Kerberos: An authentication service for open network systems. In *Proceedings of the Winter 1988 Usenix Conference*, (pp. 191–202).

Syverson, P. F., & Oorschot, P. C. V. (1996). A Unified Cryptographic Protocol Logic. In *NRL Publication*. Naval Research Lab.

Wedel, G., & Kessler, V. (1996). Formal semantics for authentication logics. In *Proceedings of ESORICS'96. LNCS*, vol. 1146 (pp. 219–241). Springer.

Wen, J., Zhang, M., & Li, X. (2005) The study on the application of ban logic in formal analysis of authentication protocols. In *Proceedings of the 7th International Conference on Electronic Commerce* (pp. 744–747). ACM Press.

Yahalom, R., Klein, B., & Beth, T. (1993). Trust relationships in secure systems—a distributed authentication perspective. In *Proceedings of the 1993 IEEE Symposium on Research in Security and Privacy* (pp. 150–164).

Ji Ma received his B.Sc degree (honours) in Computer and Information Science from the University of South Australia, Adelaide, Australia, in 2004. He is currently a PhD student at Macquarie University, Sydney, Australia. His research interests include information security, trust management, dynamic trust theory, and knowledge management.

Mehmet A. Orgun received his B.Sc. and M.Sc. degrees in computer science and engineering from Hacettepe University, Ankara, Turkey; and his PhD degree in computer science from the University of Victoria, Canada in 1991. He joined Macquarie University, Sydney, Australia in 1992 where he is currently an associate professor. His current research interests include temporal reasoning, data mining, temporal and modal logics, and reactive and distributed systems. He is a senior member of the IEEE.

A Logical Verification Method for Security Protocols Based on Linear Logic and BAN Logic

Koji Hasebe and Mitsuhiro Okada[⋆]

Department of Philosophy, Keio University
2-15-45, Mita, Minato-ku, Tokyo 108-8345 Japan
{hasebe,mitsu}@abelard.flet.keio.ac.jp

Abstract. A process following a security protocol is represented by a formal proof (of a fragment of linear logic based on the multiset rewriting model), modifying the idea by Cervesato-Durgin-Lincoln-Mitchell-Scedrov [4], while the (modified) BAN logic (which was first introduced by Burrows-Abadi-Needham [2]) is used as an evaluation semantics on security-properties for processes. By this method, we can get rid of the so called "idealization" step in the verification procedure of the BAN framework. In particular, we classify BAN-style belief-inferences into two categories; the inferences which only require some syntactic structure of a process observed by a participant on one hand, and the inferences which require a participant's knowledge on the structure of a protocol and a certain honesty assumption. We call the latter the honesty inferences. We shall show how such honesty inferences are used in the evaluation semantics for the security verification. We also point out that the evaluation inferences on freshness of nonces/keys/messages are classified as in the first category but that some of such inferences lack the information how to evaluate due to the lack of a certain concrete time-constraint setting. We introduce a natural time-constraint setting in our process/protocol descriptions and enrich the expressive power of the freshness evaluation.

1 Introduction

The purpose of this paper is to propose a formal conceptual framework and method for the security protocol study. In particular, we shall give an integration of a protocol reasoning methods based on BAN logic with a transitional framework based on the multiset rewriting model. Up to now the BAN-reasoning framework and the transitional (multiset rewriting) framework have been considered competitors rather than allieds. We shall, however, show that these two can be combined to provide an integrated framework for the

[⋆] This work was partly supported by Grants-in-Aid for Scientific Research of MEXT, Center of Excellence of MEXT on Humanity Sciences (Keio University), the Japan-US collaborative research program of JSPS-NSF, Oogata-kenkyuu-jyosei grant (Keio University) and Global Security Center grant (Keio University). The first author was also supported by Fellowship for Japanese Junior Scientists from Japan Society for the Promotion of Science.

security protocol study, where the transitional framework gives the syntactic descriptions of protocols and processes (in terms of linear logical proofs) while the BAN-reasoning framework gives the semantic evaluations for these syntactic descriptions.

We give a formal description language to represent protocols and processes, by slightly modifying the idea of Cervesato-Durgin-Lincoln-Mitchell-Scedrov [4] (also cf. [6]), where the language is considered a fragment of linear logic or, equivalently, an enriched multiset rewriting system. We represent the primitive actions of a process as formal inference rules (equivalently, multiset rewrite rules enriched with some quantifiers) and the initial states as axioms, independently of a specific protocol in question. Then, we identify a process (trace) with a formal (linear logical or rewrite) proof composed of those inference rules and axioms. A *parameterized proof* is obtained from a proof-representation of process by replacing participant names with parameterized names (which we call parameterized variables). A protocol is defined as a composition of many parameterized subproof parts (each of which is called a block; see Section 2.1 for the detail). When $P_1, \ldots, P_n$ is the list of participant variables appearing in a protocol Π, Π is also denoted as $\Pi[P_1, \ldots, P_n]$.

In the level of formal proof-representations we introduce the notion of "a process, say π, following a protocol, say Π, with respect to a participant, say A"; π follows Π with respect to A if, under a certain substitution of participant names, say A, B, ... into the variables, say P, Q, ... of $\Pi[P, Q, \ldots]$, all A's blocks can be embeddable into π. Intuitively speaking, "π follows Π with respect to A" means that π is consistent with Π from the view point of A. This notion allows us to consider a process with intruders' attack which still follows a protocol with respect to all participants other than the intruders. (See Section 2.1 for the formal definitions.)

On the other hand, we use a modified BAN logic-inference rules (initiated by Burrows-Abadi-Needham [2]) as an evaluation semantics for processes. We present the evaluation semantics related to participants' beliefs as a sort of operational semantics based on the syntax of the proof-representations of processes. We show that validity of some BAN-style belief-statements is judged purely operationally to some extent; namely, we determine validity of those basic belief-statements purely by syntactic patterns of a formal proof representing a process. Our use of the belief-statements depends on a position (location) of a proof-representation of a concrete process. In fact, such beliefs correspond to the facts actually occurring in the process. Hence we use the notion *knows*, rather than *believes* in such cases. (See Section 2.2 for the detail.)

At the same time, we investigate what kind of honesty assumptions are required in this evaluation semantics in addition. We shall formalize such belief-inferences, which we call the honesty inferences, that require some honesty assumption of a participant in the sense that the participant knows the structure of the protocol in question and that the participant presumes that the current process follows a specific protocol.

Informally, at an honesty inference you first guess where you are located in the steps of a protocol which you believe to follow, by comparing your knowledge of the structure of the protocol and the history of what you have seen and what you have sent in the past. Then you infer who sent or saw or generated what kind of things (keys, nonces, names, etc.) under the assumption that the other participants follow the protocol honestly. The formalized honesty inferences are represented in a BAN-style inference form. (See Section 3 for the honesty inferences.)

By this method, we can get rid of the so called "idealization" procedure for the security verification in the BAN framework (i.e. the validity for BAN evaluations is automatically derived from the logical structure of a proof-representation of process without any idealization procedure.) The idealization procedure has been often criticized in the literature because of its ambiguous definition (e.g. [1]), while our method clarifies, in our opinion, the BAN-logic framework from a formal point of view. We also get rid of the nested use of the belief-predicate of the original BAN-logic framework, which is a source of complexity of the original BAN-logic theory. Our method simplifies, in our opinion, the BAN-logic theory.

It is known that the expressive power of the notion of freshness in the original BAN-theory is too weak to study security conditions for some protocols (cf. Syverson-Cervesato [16]). In fact, although there is a certain criterion to evaluate (in a participant's knowledge level) freshness in some cases (such as the cases when one receives a nonce/key which you generated in the same session), there are other cases for which the current BAN-framework tells nothing about the criterion for evaluating freshness. In order to supplement the limitation of the BAN-theory we introduce a natural method of time analysis for some protocols with timestamps, by means of a linear logical representation of real time systems proposed in Kanovich-Okada-Scedrov [12]. (See Section 4 on the freshness analysis.)

In this paper, we shall use the following notations: $A, B, C, \ldots$ are used to denote specific participants' names. $A(*), B(*), C(*), \ldots$ are used for participant predicate constants, where $A(s)$ means that "the participant A has information s". $Net(s)$ means that "the message s is currently transmitted through the network". $P(*), Q(*), R(*), \ldots$ are used for participant predicate variables. On the other hand, we shall also use participant names in the term level. $A, B, C, \ldots$ are used to denote participant names and $P, Q, R, \ldots$ are used to denote participant variables in the term level. For readability we often use small letters $a, b, c, \ldots$ and $p, q, r, \ldots$ to denote $A, B, C, \ldots$ and $P, Q, R, \ldots$ in the term level.

Our language is many sorted. We use `Name`, `PublicKey`, `SecretKey`, `SharedKey` and `Nonce` as primitive sorts. We also use `Key` to denote `PublicKey` or `SecretKey` or `SharedKey`. The definition of terms is as follows: The letters $a, b, c, \ldots$ are constants of sort `Name` and the letters $p, q, r, \ldots$ are variables of sort `Name`. The capital letters $K, K', \ldots, K_1, K_2, \ldots$ and $N, N', \ldots, N_1, N_2, \ldots$ are constants of sort `Key` and of sort `Nonce`, respectively, while the small letters $k, k', \ldots, k_1, k_2, \ldots$ and $n, n', \ldots, n_1, n_2, \ldots$ are variables of the same sorts as above. Any term of `Name` or `Key` or `Nonce` is of sort `Message`. The letters $x, y, z, \ldots, x', x'', \ldots, x_1, x_2, \ldots$ are used for variables of sort `Message`. $\{s\}_K$ (the encryption of s with key K) and $\langle s_1, \ldots, s_n \rangle$ (the concatenation of messages $s_1, \ldots, s_n$) are functions of sort $\texttt{Message}\times\texttt{Key}\rightarrow\texttt{Message}$ and of sort $\texttt{Message}^n \rightarrow \texttt{Message}$, respectively. The letters $s, u, v, w, \ldots, s_1, s_2, \ldots$ are used to denote terms of the sort `Message`.

2 A Verification Method for Processes

2.1 The Processes-as-Proofs Interpretation

In this subsection, we give a method for representing a process by a (linear logical) formal proof. In Cervesato-Durgin-Mitchell-Lincoln-Scedrov [4], a specific protocol is represented by a (multiset-rewriting or linear logical) proof, where each primitive action is regarded as a rewrite rule. On the other hand, we use a notion of formal proof to represent not only a specific protocol but also a process independently of a specific protocol. For this purpose, we formalize primitive actions (and initial states, resp.) as inference rules (and axioms, resp.) more generally, independently of a specific protocol. Then we represent a stack of primitive actions as a stack of these inference rules, as a formal logical proof, independently of a specific protocol.

More precisely speaking, our language of representing a process as a proof is a Horn-fragment of first-order linear logic similarly to Cervesato-Durgin-Lincoln-Mitchell-Scedrov [4] (the formal definition is given in Appendix A). We shall use the inference form of one-sided sequents to represent a Horn-clause rule for readability. We shall extend our one-sided sequents to the two-sided sequents in Section 4 later. We represent specific participant's initial states as axioms which mean the abilities of generation of nonces or session keys or the abilities of decryption/encryption, and primitive actions as inference rules; for example, receiving or sending a message from network, encryption or decryption of a key etc. Some examples of A's axioms and of A's inference rules are as below. (See Appendix A for the complete list.)

Nonce generation:

$$\vdash \exists n A(n)$$

public key K:

$$\vdash A(K'), KP(K, K')$$

sending:

$$\frac{\vdash \Gamma, A(s)}{\vdash \Gamma, A(s), Net(s)}$$

receiving:

$$\frac{\vdash \Gamma, Net(s)}{\vdash \Gamma, A(s)}$$

encryption (public key):

$$\frac{\vdash \Gamma, KP(K, K'), A(s)}{\vdash \Gamma, KP(K, K'), A(s), A(\{s\}_K)}$$

decryption (shared key):

$$\frac{\vdash \Gamma, A(K), A(\{s\}_K)}{\vdash \Gamma, A(K), A(\{s\}_K), A(s)}$$

(In the public key axiom, predicate KP means that K and K' are key pair, and K' is secret key of A and K is a public key of A.)

We also assume the linear logical inference rules for $\otimes$, $\exists$, $\forall$ as primitive actions, too. We use "," (comma) for $\otimes$. For the $\exists$-left rule we use not only the usual $\exists$-left but also the natural deduction style $\exists$-elimination rule (to introduce the fresh constant, cf. [4]). The usual $\exists$-left rule with the left-hand side context appears only in Section 4. The resource sensitivity of the linear logical framework is used especially for describing a network state, a participant's state (and a current time state introduced in Section 4). Note that the information s of A is kept "sending" but the information s of Net is erased in "receiving".

Definition 1 A formal proof composed of some axioms for the initial states and some inference rules for the primitive actions is a *proof-representation of a process*. See Appendix A for the full list of axioms and rules. (We use the letters π, ρ in order to denote processes.) We shall identify a proof-representation with a (trace of) *process* in the rest of this paper. When we replace some participant constants, say $A, B, \ldots$ and $a, b, \ldots$, of a proof-representation of a process by variables, say $P, Q, \ldots$ and $p, q, \ldots$, we get a *parameterized proof-representation*. We call a parameterized proof-representation a *parameterized process* in this paper.

If Π is a parameterized proof-representation obtained from a proof-representation π, π is called an *instance* of Π.

We regard a position in a proof as the position of the process represented by this proof. Here $i \leq j$ means that position j is below position i in a thread of proof. We use "end" in order to denote the position of end-sequent of a proof-representation.

In our process-representation framework introduced we can define a specific protocol as a parameterized process. Before we give a formal definition of a (proof-representation of) protocol, we introduce a notion of *P's block* for some participant P. In the following definition of block, A *semi-proof* is a proof whose initial sequents are not necessarily axioms.

Definition 2 *A P's block* is a semi-proof, say Σ, satisfying the following conditions.

1. All initial sequents of Σ are axioms for P or a sequent of the form $\vdash \Gamma, Net(x)$.
2. All inferences of Σ are inference rules for P.
3. Predicate symbol $Net(*)$ doesn't appear in this part except in the initial sequent or end sequent.

We can obtain instantiated A's block by replacing participant variable of the block (say, P and p) by A and a, and we call it simply "A's block".

By means of the notion of block, regarding P's block as P's specific procedure, we define a protocol as a parameterized proof composed of some participants' blocks.

Definition 3 A *protocol*, say Π, is a parameterized process (i.e. a parameterized proof-representation) satisfying the following conditions.

1. Π is composed of some participants' blocks.
2. For each block (say Σ) in Π, all initial sequents of Σ are the axioms of Π or the end sequent of the other block in Π.
3. The end sequent of Σ is the end sequent of Π or the initial sequent of the other block in Π.

For each participant P, P's blocks appear in a protocol Π are called *P' s blocks of Π.*

The letters $\Pi, \Pi_1, \Pi_2, \ldots$ are used to denote protocols. We also use the notation $\Pi[P_1, \ldots, P_n]$ in order to indicate the list of participant variables appearing in Π, and $\Pi[A_1/P_1, \ldots, A_n/P_n]$ in order to indicate an instantiated (proof-representation of) protocol which is obtained by substituting A_i for P_i for each i ($i = 1, \ldots, n$). We use the letters $\sigma, \sigma_1, \sigma_2, \ldots$ to denote substitutions.

Here we show an example of protocol and its blocks.

Example 1 Consider the following informal description of a protocol $\Pi[P, Q]$.

1. $P \to Q : N$ (N is generated by P.)
2. $Q \to P : \{N, q\}_K$ (K is shared key with P and Q.)

According to our definitions, we represent this as a parameterized proof-representation in the left-below. This protocol is decomposed into the three blocks as shown in the right-below.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdash \exists n P(n)}{\vdash P(N)} \quad \dfrac{\vdash P(K) \quad \vdash Q(K)}{\vdash P(K),Q(K)}}{\vdash P(N),P(K),Q(K)}}{\vdash Net(N),P(K),Q(K)}}{\vdash Q(N),P(K),Q(K)} \quad \dfrac{\vdash \forall p Q(p)}{\vdash Q(q)}}{\vdash Q(N),Q(q),P(K),Q(K)}}{\vdash Q(\{N,q\}_K),P(K)}}{\vdash Net(\{N,q\}_K),P(K)}}{\vdash P(\{N,q\}_K),P(K)}}{\vdash P(N),P(q)}$$

$$\dfrac{\dfrac{\dfrac{\vdash \exists n P(n)}{\vdash P(N)} \quad \dfrac{\vdash P(K) \quad \vdash Q(K)}{\vdash P(K),Q(K)}}{\vdash P(N),P(K),Q(K)}}{\vdash Net(N),P(K),Q(K)}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdash Net(N),P(K),Q(K)}{\vdash Q(N),P(K),Q(K)}}{\vdash Q(N),Q(q),P(K),Q(K)} \quad \dfrac{\vdash \forall p Q(p)}{\vdash Q(q)}}{\vdash Q(\{N,q\}_K),P(K)}}{\vdash Net(\{N,q\}_K),P(K)}$$

$$\dfrac{\dfrac{\vdash Net(\{N,q\}_K),P(K)}{\vdash P(\{N,q\}_K),P(K)}}{\vdash P(N),P(q)}$$

In other words, we can obtain a protocol by joining the all blocks in the specific order. We call this order as *the order of blocks (of a protocol).*

In terms of the above definitions, we give a definition of *a process π following a protocol Π with respect to a participant A.*

Definition 4 For a process π and a protocol $\Pi[P, P_1, \ldots, P_n]$ and a substitution $\sigma \equiv (P := A, P_1 := B_1, \ldots, P_n := B_n)$, if π includes all A's blocks of $\Pi[A/P, B_1/P_1, \ldots, B_n/P_n]$ in the order, then π is *a process following protocol $\Pi[P, P_1, \ldots, P_n]$ with respect to A and to substitution σ.*

Example 2 According to the definition, the following proofs is a process following the protocol $\Pi[P, Q]$ (in the above example) with respect to A (and also with respect to B) and to substitution $\sigma \equiv (P := A, Q := B)$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdash \exists n A(n)}{\vdash A(N)} \quad \dfrac{\vdash A(K) \quad \vdash B(K)}{\vdash A(K), B(K)}}{\vdash A(N), A(K), B(K)}}{\vdash Net(N), A(K), B(K)}}{\vdash I(N), A(K), B(K)}}{\vdash Net(N), A(K), B(K)}}{\vdash B(N), A(K), B(K)} \quad \dfrac{\vdash \forall p B(p)}{\vdash B(b)}}{\vdash B(N), B(b), A(K), B(K)}}{\vdash B(\{N, b\}_K), A(K)}}{\vdash Net(\{N, b\}_K), A(K)}}{\vdash A(\{N, b\}_K), A(K)}}{\vdash A(N), A(b)}$$

2.2 The BAN Logic as an Evaluation Semantics about Processes

In this subsection, we introduce an evaluation method about each participant's belief at a position in a process. For this evaluation we use a modified BAN logic or its successors (e.g. Gong-Needham-Yahalom [9], Syverson-van Oorschot [15]), and regard this as some kind of semantics about belief for a process. The formal definition of our evaluation framework is given in Appendix B.

Our evaluation method is different from the original BAN logic framework. The first point is that our method is to evaluate processes, while the original BAN logic is to evaluate protocols. The second point is that we derive belief-statements about each participant's belief from a formal proof representing a process rather than an informal description of a protocol-process. The third point is that we evaluate a belief-statement relative to a position in a process; we use the belief-statement of the form $A|\!\equiv_i \varphi$ ("A believes φ at position i") rather than $A|\!\equiv \varphi$. (We omit a position index if we understand which position is intended clearly in the context.)

The features of our evaluation method are listed below:

– We get rid of the "idealization" procedures from the evaluation.
– We get rid of any nested use of the belief predicate (such as $A|\!\equiv_i B|\!\equiv_j \varphi$).

In our method, a process is evaluated by the following procedure: First we derive some statements about each participant's belief by the "evaluation criteria" listed below. Next by means of our BAN-style belief-inferences (see Appendix B) we derive new beliefs from these statements.

Before we introduce the evaluation criteria, we introduce some notions for preparation.

Definition 5 (Decomposed subterm) We define a decomposed subterm of a term t as follows:

1. t is a decomposed subterm of t.
2. If t is of the form $\langle s_1, \ldots, s_n \rangle$, and if s is a decomposed subterm of s_i (for some $i \leq n$), then s is a decomposed subterm of t.

Definition 6 (accessible to A) For a sequent $\vdash \Gamma$ and for a participant predicate A, a term s is accessible to A iff there exists a term t such that $A(t)$ appears in Γ and that s is a decomposed subterm of t.

Now we introduce the evaluation criteria.

The Evaluation Criteria

1. **Evaluation criterion for $A |\!\!\equiv^{\pi}_{i} \overset{K}{\mapsto} B$ and $A |\!\!\equiv^{\pi}_{i} KP(K, K')$:**
   If the following sequent
   $$\vdash B(K'), KP(K, K')$$
   appears in a proof π as an axiom for some B, then the statements $A |\!\!\equiv^{\pi}_{i} \overset{K}{\mapsto} B$ and $A |\!\!\equiv^{\pi}_{i} KP(K, K')$ are evaluated as a valid statements for some A at any position i in the proof π.

2. **Evaluation criterion for $A |\!\!\equiv^{\pi}_{i} A \overset{K}{\leftrightarrow} B$**
   If the following sequent
   $$\vdash A(K), B(K)$$
   appears in a proof π as an axiom for some A and B, then $A |\!\!\equiv^{\pi}_{i} A \overset{K}{\leftrightarrow} B$ is evaluated as a valid statement at any position i in the proof π.

3. **Evaluation criterion for $A |\!\!\equiv^{\pi}_{i} Gen(A, s)$:**
   If the following proof structure
   $$\frac{\vdash \Gamma, \exists x A(x)}{\vdash \Gamma, A(s)}\ \exists\text{-elimination}$$
   appears in a proof π such that the lower sequent has position j for some A, then $A |\!\!\equiv^{\pi}_{i} Gen(A, s)$ is evaluated as a valid statement at any position $i \geq j$ in the proof π (where the statement $Gen(A, s)$ means that "s is generated by A").

4. **Evaluation criterion for $A{\models}_i^\pi \exists y_1 \cdots y_n(A \lhd s)$:**
Consider the following proof structure.

$$\cfrac{\cfrac{\vdash \Gamma_1, Net(s_1)}{\vdash \Gamma_1, A(s_1)}\ \text{receiving}}{\vdash \Gamma_2, A(s_2)}\ \text{decryption rules and concatenation rules}$$

Here $\vdash \Gamma_2, A(s_2)$ is derived from $\vdash \Gamma_1, A(s_1)$ by only some decryption rules. (We can neglect concatenation rules. See Appendix A.) We assume that the sequent $\vdash \Gamma_2, A(s_2)$ has position j.
If the above proof structure appears in a proof π for some A, then $A{\models}_i^\pi \exists y_1 \cdots y_n(A \lhd s)$ is evaluated as a valid statement at any position $i \geq j$ in the proof π, where s is obtained from s_1 by replacing each subterm u_l which is not accessible to A in $\vdash \Gamma_2, A(s_2)$ $(l = 1, \ldots, n)$ with y_l (cf. see Example 3 below).

Remark. Here by using $\exists y_1 \cdots y_n$ we express that the bound variables $y_1 \cdots y_n$ are messages of which A doesn't know the actual values, because A cannot see the contents of the encrypted messages.

5. **Evaluation criterion for $A{\models}_i^\pi \exists y_1 \cdots y_n(B \mid\!\sim s)$:**
Consider the following proof structure.

$$\cfrac{\cfrac{\cfrac{\vdash \Gamma_1, Net(s_1)}{\vdash \Gamma_1, A(s_1)}\ \text{receiving}}{\vdash \Gamma_2, A(s_2)}\ \text{decryption rules and concatenation rules}}{\vdash \Gamma_3, A(s_3)}\ \text{decryption with } K$$

Here $\vdash \Gamma_2, A(s_2)$ is derived from $\vdash \Gamma_1, A(s_1)$ by only some decryption rules, and $\vdash \Gamma_3, A(s_3)$ is derived from $\Gamma_2, A(s_2)$ by decryption rule about the key K. The sequent $\vdash \Gamma_3, A(s_3)$ has position j.
If the above proof structure appears in a proof for some A, and if either the statement $A{\models}_j^\pi A \overset{K}{\leftrightarrow} B$ is valid or $A{\models}_j^\pi \overset{K'}{\mapsto} B$ and $A{\models}_j^\pi KP(K', K)$ is valid for some B, then the statement $A{\models}_i^\pi \exists y_1 \cdots y_n(B \mid\!\sim s)$ is evaluated as a valid statement at any position $i \geq j$, where s is obtained from s_3 by replacing each subterm u_l which is not accessible to A in $\vdash \Gamma_3, A(s_3)$ $(l = 1, \ldots, n)$ with y_l.

Example 3 We show an example about the criteria 4 and 5. If the following proof structure appears in a proof π;

$$\cfrac{\cfrac{\cfrac{\vdash \Gamma, Net(\{\{N_1, a\}_{K_2}, N_2\}_{K_3})}{\vdash \Gamma, A(\{\{N_1, a\}_{K_2}, N_2\}_{K_3})}\ \text{receiving}}{\vdash \Gamma', A(\langle\{N_1, a\}_{K_2}, N_2\rangle)}\ \text{decryption with } K_3}{\vdash \Gamma'', A(\langle\langle N_1, a\rangle, N_2\rangle)}\ \text{decryption with } K_2$$

where we assume the positions of lower sequent of receiving rule and the next two sequents are i, $i+1$ and $i+2$, respectively. Then we evaluate $A|\!\equiv^{\pi}_{j_1} \exists y(A \triangleleft \{y\}_{K_3})$, $A|\!\equiv^{\pi}_{j_2} \exists y(A \triangleleft \{\{y\}_{K_2}, N_2\}_{K_3})$ and $A|\!\equiv^{\pi}_{j_3} A \triangleleft \{\{N_1, a\}_{K_2}, N_2\}_{K_3}$ (where $j_1 \geq i$, $j_2 \geq i+1$ and $j_3 \geq i+2$) as a valid statements. Moreover, if $A|\!\equiv^{\pi}_{i+1} A \overset{K_3}{\leftrightarrow} B$, then we also evaluate $A|\!\equiv^{\pi}_{j_2} \exists y(B|\!\sim \langle \{y\}_{K_1}, N_2\rangle)$ as a valid statement, and if $A|\!\equiv^{\pi}_{i+2} \overset{K_1}{\mapsto} B$ and $A|\!\equiv^{\pi}_{i+2} KP(K_1, K_2)$, then $A|\!\equiv^{\pi}_{j_3} B|\!\sim \langle N_1, a\rangle$ is evaluated as a valid statement.

6. **Evaluation criterion for $A|\!\equiv^{\pi}_{i} \exists y_1 \cdots y_n(A|\overset{*}{\sim} s)$:**
   Consider the following proof structure.

$$\dfrac{\dfrac{\vdash \Gamma_1, A(s_1)}{\vdash \Gamma_2, A(s_2)}\text{ encryption rules and concatenation rules}}{\vdash \Gamma_2, Net(s_2), A(s_2)}\text{ sending}$$

   Here $\vdash \Gamma_2, A(s_2)$ is derived from $\vdash \Gamma_1, A(s_1)$ by only some encryption rules (where we can neglect concatenation rules), and the sequent $\vdash \Gamma_2, Net(s_2), A(s_2)$ has position j.
   If the above proof structure appears in a proof π for some A, then $A|\!\equiv^{\pi}_{i} \exists y_1 \cdots y_n(A|\overset{*}{\sim} s)$ is evaluated as a valid statement at any position $i \geq j$, where s is obtained from s_2 by replacing each subterm u_l which is not accessible to A in $\vdash \Gamma_2, Net(s_2), A(s_2)$ $(l = 1, \ldots, n)$ by y_l.

7. **Evaluation criterion for $A|\!\equiv^{\pi}_{i} A \ni s$**
   If a predicate of the form $A(s)$ appears in a proof π for some A at a position j in π, then $A|\!\equiv^{\pi}_{i} A \ni s$ is evaluated as a valid statement at any position $i \geq j$.

We listed above seven evaluation criteria. By the definition of the criteria, belief-statements derived from the evaluation criteria correspond to the facts actually occurring in the process. Hence, we consider such belief-statements as knowledge rather than belief and we call them *knowledge-statements.* We introduce a notation $A \models^{\pi}_{i} \varphi$ which represents "A knows φ at position i in π" and we define if $A \models^{\pi}_{i} \varphi$ then $A|\!\equiv^{\pi}_{i} \varphi$ for any A and φ and π. These rules are not enough for belief-inferences. We shall discuss what rules are needed to evaluate such processes and introduce these rules in Section 3.

3 Honesty Inferences and the Use of Belief Histories

3.1 Introduction of Honesty Inferences

In Section 2.2, we introduced some evaluation criteria of participant's belief independently of the participant's knowledge of a protocol (and we treated such the belief as knowledge). However, these criteria are not strong enough for our evaluation semantics. In order to capture stronger belief-inferences, the original BAN logic needed to interpret a protocol (this interpretation is called "idealization") informally. In our framework, instead of such "idealization", we introduce "honesty inferences" into our evaluation framework. These inference rules assume each participant's knowledge about the protocol and his/her belief of the other participants' honesty, (i.e. the other participants faithfully follow the protocol procedure).

First we show an example of an evaluation for a process following a protocol. Here we consider the following protocol $\Pi[P, Q, R]$ which intends to share the key K (created by R) between P and Q, and evaluate a *standard process* following the protocol $\Pi[P, Q, R]$ (which means that the process follows the protocol $\Pi[P, Q, R]$ with respect to A, B and C and substitution $P := A$, $Q := B$ and $R := C$, and does not include any other inferences).

Example 4

(1) An informal description of a protocol $\Pi[P, Q, R]$

1. $P \to R : \{p, q, N_P\}_{K_R}$
2. $R \to Q : \{N_P, p, K\}_{K_Q}, \{N_P\}_K$
3. $Q \to P : \{N_P, q, K\}_{K_P}$
4. $P \to Q : \{N_P, p, K\}_{K_Q}, \{N_P\}_K$

(2) An informal description of a standard process $\pi \equiv \Pi[A/P, B/Q, C/R]$

1. $A \to C : \{a, b, N_A\}_{K_C}$
2. $C \to B : \{N_A, a, K\}_{K_B}, \{N_A\}_K \cdots$ position i
3. $B \to A : \{N_A, b, K\}_{K_A}$
4. $A \to B : \{N_A, a, K\}_{K_B}, \{N_A\}_K \cdots$ position j

(Here we use some abbreviations: in the informal protocol/process-descriptions N_P represents a nonce generated by P with nonce generation rule beforehand, and K_P, K_Q and K_R represent public keys for P, Q, and R, respectively, introduced by suitable axioms for keys beforehand. We also use corresponding abbreviation of some belief-statements: generally $A|\!\equiv_i B|\!\sim N_C$ is abbreviation of "$A|\!\equiv_i B|\!\sim N$ and $A|\!\equiv_i Gen(C, N)$", and $A|\!\equiv_i B \lhd \{s\}_{K_C}$ is abbreviation of "$A|\!\equiv_i B \lhd \{s\}_K$ and $A|\!\equiv_i \overset{K}{\mapsto} C$", and so on.)

In (2) of Example 4 above we consider an evaluation about B's belief at the position where B decrypts the key K_B and K on the line 2 in (2). We call this position as i.

By means of the evaluation criteria of Section 2.2, we can derive the statement $B \models_i^\pi B \lhd \langle\langle N_A, a, K\rangle, N_A\rangle$. (Namely, B knows that B receives $\langle\langle N_A, a, K\rangle, N_A\rangle$.) However, by means of only the evaluation criteria of Section 2.2 and belief-inferences in Appendix B, B cannot infer who sent this message or who had key K. Generally in a process following a protocol, when a participant receives a message, he/she cannot always know who has sent the message only by the information of the message. However, in the case of Example 4, if B knows the whole structure of the protocol $\Pi[P, Q, R]$ and assumes the other participants' honesty (i.e., A and C follow the protocol $\Pi[P, Q, R]$ with respect to $P := A$, $Q := B$ and $R := C$), and if B knows that B actually follows the protocol (i.e., that B received the message $\langle\{N_A, a, K\}_{K_B}, \{N_A\}_K\rangle$, which is consistent with the process $\pi \equiv \Pi[A/P, B/Q, C/R]$), then B can believe that the message $\langle\{N_A, a, K\}_{K_B}, \{N_A\}_K\rangle$ was sent by C at position i. This is because if A, B and C follow the protocol $\Pi[P, Q, R]$ with respect to $P := A$, $Q := B$ and

$R := C$, then C should send the message $\langle \{N_A, a, K\}_{K_B}, \{N_A\}_K \rangle$, which is actually received by B.

We represent this B's inference as follows.

$$\frac{B \models_i^{\pi} B \lhd \langle \{N_A, a, K\}_{K_B}, \{N_A\}_K \rangle}{B {\models}_i^{\pi, \Pi}\ C|\overset{*}{\sim} \langle \{N_A, a, K\}_{K_B}, \{N_A\}_K \rangle\ [A/P, B/Q, C/R]}$$

In an inference rule of this type, the participant's knowledges of the history of his/her own sending and receiving actions are listed as the upper statements (premises). We call these premises his/her *history*. (Each statement of the history is derived by only the evaluation criteria introduced in Section 2.2.) The lower statement (conclusion) of the above inference rule is B's belief based on his/her knowledge about the whole structure of the protocol $\Pi[P, Q, R]$ and on his/her assumption that the other participants follow the protocol $\Pi[P, Q, R]$ with respect to $P := A$, $Q := B$ and $R := C$ (namely he/she assumes that the process $\pi \equiv \Pi[A/P, B/Q, C/R]$ is realized). We call the part $[A/P, B/Q, C/R]$ an *assignment*. (Generally, for any participant constant A and for any protocol $\Pi[P_1, \ldots, P_n]$ and for any position i of a given process, $A|\equiv_i^{\pi, \Pi}\ \varphi[B_1/P_1, \ldots, B_n/P_n]$ means that A believes φ at position i in π under the interpretation of P_m by B_m for each $m \leq n$. We call the part $[B_1/P_1, \ldots, B_n/P_n]$ as an assignment. The original notation $A|\equiv_i\ \varphi$ is interpreted as "$A|\equiv_i\ \varphi[B_1/P_1, \ldots, B_n/P_n]$ for any $B_1, \ldots, B_n$". However, we shall omit the assignment part $[B_1/P_1, \ldots, B_n/P_n]$ if it is clear from the context.)

Here we point out what is the role of a participant's history. The role of the history is to detect a position in the protocol. For example, in the process (2) of Example 4 above, if B ignores his/her history when B receives the message $\langle \{N_A, a, K\}_{K_B}, \{N_A\}_K \rangle$, then the position cannot be determined and B cannot detect whether A or C sent the message, because B receives the same messages twice in the process. However, if B's history is $B \models_j^{\pi} B \lhd \langle \{N_A, a, K\}_{K_B}, \{N_A\}_K \rangle$, $B \models_j^{\pi} B|\overset{*}{\sim} \{N_A, b, K\}_{K_A}$ and $B \models_j^{\pi} B \lhd \langle \{N_A, a, K\}_{K_B}, \{N_A\}_K \rangle$, the current position can be determined as j (where B receives the second $\langle \{N_A, a, K\}_{K_B}, \{N_A\}_K \rangle$) and B can detect that the message is sent from A. This inference is represented as follows.

$$\frac{\begin{array}{c} B \models_j^{\pi} B \lhd \langle \{N_A, a, K\}_{K_B}, \{N_A\}_K \rangle \qquad B \models_j^{\pi} B|\overset{*}{\sim} \{N_A, b, K\}_{K_A} \\ B \models_j^{\pi} B \lhd \langle \{N_A, a, K\}_{K_B}, \{N_A\}_K \rangle \end{array}}{B {\models}_j^{\pi, \Pi}\ A|\overset{*}{\sim} \{N_A, b, K\}_{K_B}\ [A/P, B/Q, C/R]}$$

We introduce this kind of inference rules as *honesty inferences*. (Formal definition is given in the next subsection.) Using these inferences we evaluate each participant's belief in a process following protocol. Moreover, when we derive some participant's belief about some secrecy properties (a goal of this protocol) however by using some honesty inferences, then the usage of these inferences serves as a detection of the flaw points in a protocol.

3.2 Formal Definition of the Honesty Inferences

Before defining the honesty inferences, we introduce some notions for preparation.

Definition 7 Consider the following P's block.

$$\begin{array}{c}
\vdash \Gamma, Net(s) \\
(\alpha)\left\{\begin{array}{c} \hline\hline \vdash \Gamma', P(s') \\ \hline \vdash \Gamma'', P(s'') \end{array}\right. \text{lowest decryption rule} \\
\vdots \\
(\beta)\left\{\begin{array}{c} \vdash \Delta'', P(u') \\ \hline \vdash \Delta', P(u) \end{array}\right. \text{highest encryption rule} \\
\hline\hline
\vdash \Delta, Net(u)
\end{array}$$

The part (α) includes some decryption rules and concatenation rules, and any decryption rule doesn't appear in the lower part of the sequent $\vdash \Gamma'', P(s'')$. The part (β) includes some encryption rules and concatenation rules, and any encryption rule doesn't appear in the upper part of $\vdash \Delta'', P(u')$. Here we call the part (α) as *decryption part of the block* and the part (β) as *encryption part of the block*. (We point out that it is possible that $\vdash \Delta'', P(u')$ appears above $\vdash \Gamma'', P(s'')$.)

If we consider an A's block for any participant constant A, we also define same as above.

From now we give the formal definition of the honesty inferences.

We consider an arbitrary protocol $\Pi[P, Q, R_1, \ldots, R_m]$ and an instance $\Pi[A/P, B/Q, C_1/R_1, \ldots, C_m/R_m]$ for some participant constants A, B and $C_1, \ldots, C_m$. In the following discussion we omit $R_1, \ldots, R_m$ and $C_1, \ldots, C_m$, and we abbreviate $\Pi[A/P, B/Q, C_1/R_1, \ldots, C_m/R_m]$ as $\Pi[A/P, B/Q]$.

Consider the following part of $\Pi[A/P, B/Q]$ (it is called π).

$$\begin{array}{c}
\textbf{A part of } \Pi[A/P, B/Q] \\
\left.\begin{array}{c}
\ddots\ \vdots\ \iddots \\
\vdots \\
\dfrac{\vdash \Gamma, A(s_{n-1})}{\vdash \Gamma, Net(s_{n-1})}\ \text{sending}
\end{array}\right\}(\alpha) \\
\vdots \\
\boxed{B\text{'s block}}\ \cdots\cdots\ (\beta) \\
\vdots \\
\dfrac{\vdash \Gamma', Net(s_n)}{\vdash \Gamma', A(s_n)}\ \text{receiving}
\end{array}$$

Assume that π satisfies the following conditions.

1. The part (α) in π is the initial part of a protocol $\Pi[A/P, B/Q]$, which includes A's $n-1$ blocks $(n \geq 1)$.
2. For each A's k-th block $(k = 1, \ldots, n)$, the sequents $\vdash \Gamma_k, Net(s_k)$ and $\vdash \Delta_k, Net(u_k)$ appear in the top and bottom of the block respectively. (If A's first block is a first block of Π, $\vdash \Gamma_1, Net(s_1)$ doesn't appear in the A's first block.)
3. Between A's $n-1$-th block and n-th block, at least one B's block (β).

Honesty Inferences (1)

If the following B's block (β) appears in π,

$$\begin{array}{rl}
 & \vdash \Phi, Net(v_1) \\
\text{(decryption part)} \left\{ \begin{array}{c} \vdots \\ \hline \vdash \Phi', B(v'_1) \end{array} \right. & \text{lowest decryption rule} \\
 & \vdots \\
\text{(encryption part)} \left\{ \begin{array}{c} \vdash \Psi', B(v'_2) \\ \hline \vdots \end{array} \right. & \text{highest encryption rule} \\
 & \overline{\vdash \Psi, Net(v_2)}
\end{array}$$

then we can use the following inference rules for any position i.

$$\frac{\begin{array}{c} A \models^{\pi}_{i} \exists \boldsymbol{y}(A \lhd s'_1) \quad A \models^{\pi}_{i} \exists \boldsymbol{y'}(A \overset{*}{\mid\!\sim} u'_1) \cdots A \models^{\pi}_{i} \exists \boldsymbol{y}(A \lhd s'_{n-1}) \\ A \models^{\pi}_{i} \exists \boldsymbol{y'}(A \overset{*}{\mid\!\sim} u'_{n-1}) \quad A \models^{\pi}_{i} \exists \boldsymbol{y}(A \lhd s'_n) \end{array}}{A \mathrel{|}\!\!\equiv^{\pi,\Pi}_{i} \exists nk(B \lhd v_3)\ [A/P, B/Q]}$$

$$\frac{\begin{array}{c} A \models^{\pi}_{i} \exists \boldsymbol{y}(A \lhd s'_1) \quad A \models^{\pi}_{i} \exists \boldsymbol{y'}(A \overset{*}{\mid\!\sim} u'_1) \cdots A \models^{\pi}_{i} \exists \boldsymbol{y}(A \lhd s'_{n-1}) \\ A \models^{\pi}_{i} \exists \boldsymbol{y'}(A \overset{*}{\mid\!\sim} u'_{n-1}) \quad A \models^{\pi}_{i} \exists \boldsymbol{y}(A \lhd s'_n) \end{array}}{A \mathrel{|}\!\!\equiv^{\pi,\Pi}_{i} \exists nk(B \overset{*}{\mid\!\sim} v_4)\ [A/P, B/Q]}$$

Here for each s'_k for $k = 1, \ldots, n$ is obtained from s_k by replacing each subterm w_l for $l = 1, \ldots, m$ which is not accessible to A in the decryption part with y_l. Also for each u'_k for $k = 1, \ldots, n-1$ is obtained form u_k by replacing each subterm $w'_{l'}$ for $l' = 1, \ldots, m'$ which is not accessible to A in the encryption part with $y'_{l'}$. v_3 (v_4, resp.) is obtained from v_1 (v_2, resp.) by replacing each constants N and K which are not accessible to A at position i with variables n and k.

Honesty Inference (2)

If the following B's block (β) appears in π,

$$\cfrac{\cfrac{\cfrac{\begin{array}{c}\vdots\\ \vdash \Phi', B(v_1), B(v_2)\end{array}}{\vdash \Phi', B(\langle v_1, v_2\rangle)}\text{ concatenation}}{\vdash \Phi, B(w)}\text{ encryption rules and concatenation rules}}{\vdash \Phi, Net(w)}\text{ sending}$$

(where the term w is obtained from $\langle v_1, v_2\rangle$ by some encryption rules and concatenation rules), then we can use the following inference rule at any position i.

$$\frac{\begin{array}{c}A \models_i^{\pi} \exists \boldsymbol{y}(A \triangleleft s_1') \quad A \models_i^{\pi} \exists \boldsymbol{y'}(A \overset{*}{\mid\!\sim} u_1') \cdots A \models_i^{\pi} \exists \boldsymbol{y}(A \triangleleft s_{n-1}') \quad A \models_i^{\pi} \exists \boldsymbol{y}(A \overset{*}{\mid\!\sim} u_{n-1}')\\ A \models_i^{\pi} \exists nk(B \overset{*}{\mid\!\sim} v_1) \quad A \models_i^{\pi} \exists \boldsymbol{y}(A \triangleleft s_n')\end{array}}{A \models_i^{\pi, \Pi} \exists nk(B \triangleleft \langle v_1, v_2'\rangle)\ [A/P, B/Q]}$$

Here for each s_k' and u_k' $(k = 1, \ldots, n)$ is the same as the definition of honesty inference (1). And v_2' is obtained from v_2 by replacing each constants N, K with n and k where each N and K is not accessible to A at position i.

Honesty Inference (3)

If the following B's block (β) appears in π,

$$\cfrac{\cfrac{\cfrac{\begin{array}{c}\vdots\\ \vdash \Phi', B(u)\end{array}}{\vdash \Phi', B(\{u\}_K)}\text{ encryption rule}}{\vdash \Phi', B(v)}\text{ encryption rules and concatenation rules}}{\vdash \Phi, Net(v)}\text{ sending}$$

then we can use the following inference rule at any position i.

$$\frac{\begin{array}{c}A \models_i^{\pi} \exists \boldsymbol{y}(A \triangleleft s_1') \quad A \models_i^{\pi} \exists \boldsymbol{y'}(A \overset{*}{\mid\!\sim} u_1') \cdots \quad A \models_i^{\pi} \exists \boldsymbol{y}(A \triangleleft s_{n-1}') \quad A \models_i^{\pi} \exists \boldsymbol{y'}(A \overset{*}{\mid\!\sim} u_{n-1}')\\ A \models_i^{\pi} \exists nk(B \overset{*}{\mid\!\sim} \{u\}_K) \quad A \models_i^{\pi} \exists \boldsymbol{y}(A \triangleleft s_n')\end{array}}{A \models_i^{\pi, \Pi} \exists nk(B \triangleleft u)\ [A/P, B/Q]}$$

Here for each s_k' and u_k' $(k = 1, \ldots, n)$ is is same as the definition of honesty inference (1) and (2).

Remark on a statement about secrecy. We don't define an inference rule for deriving a statement about secrecy directly (as the original BAN logic did). However, we treat $A| \equiv A \overset{s}{\rightleftharpoons} B$ for some A and B and for some s as valid statement if the evaluation criterion 8 below is satisfied at the end position. Before we state the evaluation criterion 8, we introduce a notion *s is preserved to A and B with respect to t* for preparation.

Definition 8 *s is preserved to A and B with respect to t* if the following conditions.

1. If t is of the form $\{s'\}_{K_A}$ or $\{s'\}_{K_B}$ or $\{s'\}_{K_{AB}}$ and if s is decomposed subterm of s', then s is preserved to A and B with respect to t.
2. If t is not subterm of s, then s is preserved A and B with respect to t.
3. If t is of the form $\langle s_1, \ldots, s_n \rangle$ and if s is preserved to A and B with respect to s_i for all i, then s is preserved to A and B with respect to t.
4. If t is of the form $\{s'\}_{K_A}$ or $\{s'\}_{K_B}$ or $\{s'\}_{K_{AB}}$ and if s is preserved to A and B with respect to s', then s is preserved to A and B with respect to t.

8. **Evaluation criterion for $A| \!\equiv^{\pi,\Pi}_{end} A \overset{s}{\rightleftharpoons} B$:**

If the following conditions are satisfied;

1. $A| \!\equiv^{\pi,\Pi}_{end} A \ni s$ and $A| \!\equiv^{\pi,\Pi}_{end} B \ni s$ are valid for a protocol Π and a process $\pi \equiv \Pi[A/P, B/Q]$
2. For each $\vdash \Gamma, Net(t)$, appearing in $\pi \equiv \Pi[A/P, B/Q]$, s is preserved to A and B with respect to t.

then $A| \!\equiv^{\pi,\Pi}_{end} A \overset{s}{\rightleftharpoons} B$ is evaluated as a valid statement at the end position.

We have shown the criteria for validity of knowledge-statements in Section 2.2 and additional reasoning (honesty inferences) in this section. Using these evaluation frameworks, however, we cannot detect some kind of replay attacks, like an attack on the BAN-Yahalom protocol [2]. According to Syverson-Cervesato[16], these replay attacks cannot be detected in the frameworks based on the original BAN logic because in such frameworks freshness is very weak mechanism only available to distinguish a current run from another. In fact, consider the following belief-inferences based on the original BAN logic.

$$\dfrac{\dfrac{A| \!\equiv B |\!\sim \langle N, K\rangle \qquad \dfrac{A| \!\equiv \sharp(N)}{A| \!\equiv \sharp(\langle N, K\rangle)}}{A| \!\equiv B \overset{*}{|\!\sim} \langle N, K\rangle}}{A| \!\equiv B \ni K}$$

We can observe that these inferences only tell us the need of "freshness of nonce N" to derive $A| \!\equiv B \ni K$, and tell us nothing as to when this freshness is guaranteed. One may think this problem would be solved with introducing timestamps. However even if we replace the nonce by a timestamp it makes no difference, because one would get exactly the same BAN-inferences as before with a timestamp to rather than a nonce N and it would only tell us the need of $A| \!\equiv \sharp(t)$, namely the freshness of t, instead of freshness of N.

Therefore we would like to have to make the freshness notion more powerful and more expressive in order to evaluate the validity of $A| \!\equiv B \ni K$. For this purpose, in the next section, we shall introduce an evaluation criterion for freshness in terms of the real-time notion and its quantitative time-constraints.

4 Improvement of Evaluation Criterion for Freshness

For the purpose of improving freshness mechanism, in Section 4.1 we add the real-time notion and its quantitative time constraints to our process-representation framework. Our method for the extension is following Kanovich-Okada-Scedrov[12]. In this extended framework, lifetime of a message with timestamp is defined by upper-bound time-constraint (i.e., lifetime is defined as D-time). Then we can introduce a natural evaluation criterion such that "a message is defined to be fresh within D-time" in the evaluation framework. In Section 4.2, we shall show how to evaluate some participants' knowledge about freshness which depends on lifetime of a message in the extended framework.

4.1 Definition of the Notion of Freshness Using Time-Constraint

We extend the language and inference rules (introduced in Section 2.1) as follows.

Sorts: `Time`

Terms: Any real number is a term of sort `Time`. $t, t', \ldots, t_1, t_2, \ldots, H, H_1, H_2, \ldots$, $D, D_1, D_2, \ldots$ are variables of sort `Time`. If t_1 and t_2 are terms of sort `Time` and s is a term of `Message`, then $u(\langle s, t_1 \rangle, t_2)$ is also a term of `Message`. (Here u is a function symbol connecting s_1, t_1 and t_2.)

Predicate symbols: $Time(t), L(t)$ where t is a term of sort `Time`.

Inference rules: we add the *tick axiom* which means the time progression.

Tick:

$$\frac{\Gamma \vdash \Delta, Time(t)}{t < t', \Gamma \vdash \Delta, Time(t')}$$

We set an upper-bound time constraint to a message with timestamp by the next two rules.

Timestamp:

setting rule for A:

$$\frac{\Gamma \vdash \Delta, A(s), Time(t)}{\Gamma \vdash \Delta, A(u(\langle s, t \rangle, D), Time(t))}$$

discarding rule for A:

$$\frac{\Gamma \vdash \Delta, A(u(\langle s, t \rangle, D)), Time(t')}{t' < t + D, \Gamma \vdash \Delta, A(\langle s, t \rangle), Time(t')}$$

The setting rule means that A adds the timestamp t to message s with time-constraint D. The discarding rule means that A checks freshness of the message s by timestamp t with D, and the time-constraint condition appear in the left-hand side of the lower sequent. Here the term $u(\langle s, t \rangle, D)$ plays a role for the upper-bound timer for lifetime of the message s. (u is a function symbol connecting the terms s, t and D.)

On the other hand, we introduce the following evaluation criterion for freshness into our evaluation semantics.

9. **Evaluation criterion for $A \models_i^\pi \sharp(\langle s, t\rangle)$:**

   If there is a following proof structure in a proof π;

$$\frac{\Gamma \vdash \Delta, A(u(\langle s,t\rangle, D), Time(t'))}{t' < t + D, \Gamma \vdash \Delta, A(\langle s,t\rangle), Time(t')}$$

   and the left hand side of the lower sequent $t' < t + D$ is valid, then $A \models_i^\pi \sharp(\langle s, t\rangle)$ is evaluated as a valid statement at any position i such that its time is before t'.

We point out that a statement derived by this evaluation criterion is A's knowledge rather than belief (cf. the end of Section 2.2). This criterion means that a message is fresh when the time-constraint condition $t' < t + D$ is satisfied.

4.2 Evaluation of Knowledge about Freshness

In this subsection, we show an evaluation in our extended evaluation framework introduced in the previous subsection. As an example, we introduce here the modified BAN-Yahalom protocol as follows (which is obtained by adding timestamps to the BAN-Yahalom protocol), and show how to evaluate some participant's knowledge about freshness which depends on lifetime of a message.

An Informal Description of the Modified BAN-Yahalom Protocol

1. $P \to Q : p, N_P$
2. $Q \to R : q, N_Q, \{p, N_P, t_Q\}_{K_{QR}}$
3. $R \to P : N_Q, \{q, K_{PQ}, N_P, t_R\}_{K_{PR}}, \{p, N_Q, K_{PQ}, t_R\}_{K_{QR}}$
4. $P \to Q : \{p, N_Q, K_{PQ}, t_R\}_{K_{QR}}, \{N_Q, t_P\}_{K_{PQ}}$

(Here t_Q, t_R and t_P are timestamps generated by timestamp-setting rules for P (at position 4), Q (at position 2) and R (at position 3), respectively. We set the upper-bound constraints to the freshness of the messages with timestamps t_Q, t_R and t_P as D_1 (for t_Q), D_2 (for t_R checked by P), D_3 (for t_R checked by Q) and D_4 (for t_P), respectively.)

We introduce the lower-bound time constraint to the handling duration of messages by the following two rules (which are modified sending and receiving rules introduced in Section 2.1). These rules mean that it takes more than or equal to H time to handle the message.

sending:

$$\frac{\Gamma \vdash \Delta, A(s), L(t+H), Time(t')}{t + H \le t', \Gamma \vdash \Delta, A(s), Net(s), Time(t')}$$

receiving:

$$\frac{\Gamma \vdash \Delta, Net(s), Time(t)}{\Gamma \vdash \Delta, A(s), L(t+H), Time(t)}$$

For this protocol, we can consider the following two processes: process π_1 is a standard process following this protocol with respect to $P := B$, $Q := A$ and $R := S$, and process π_2 is a process in which A and S follow this protocol and which includes C's attack (due essentially to Syverson[14]). In these processes, the position where S checks freshness of $\langle b, N_1, t_A \rangle$ is indicated as i, and we assume that this checking at position i happens at time t'.

Process π_1 (A standard process following the modified BAN-Yahalom protocol with respect to $P := B$, $Q := A$ and $R := S$)

1. $B \to A : b, N_1$
2. $A \to S : a, N_2, \{b, N_1, t_A\}_{K_1}$ $\cdots\cdots$ position i, time t'
3. $S \to B : N_2, \{a, K, N_1, t_S\}_{K_2}, \{b, N_2, K, t_S\}_{K_1}$
4. $B \to A : \{b, N_2, K, t_S\}_{K_1}, \{N_2, t_B\}_K$

(Here N_1 and N_2 are generated by B and A, respectively. K_1 and K_2 are shared keys for A with S and B with S, respectively, and K is session key generated by S for B with A. t_A, t_S and t_B are timestamps generated by timestamp-setting rules for A, S and B respectively.)

Process π_2 (An attacked process on the BAN-Yahalom protocol)

1. $A \to C(B) : a, N_1$
   I. $C(B) \to A : b, N_1$
   II. $A \to C(S) : a, N_2, \{b, N_1, t_A\}_{K_1}$
      ii. $C(A) \to S : a, N_1, \{b, N_1, t_A\}_{K_1}$ $\cdots\cdots$ position i, time t'
      iii. $S \to C(B) : N_1, \{a, K, N_1, t_S\}_{K_2}, \{b, K, N_1, t_S\}_{K_1}$
3. $C(S) \to A : N_3, \{b, K, N_1, t_S\}_{K_1}, \{a, K, N_1, t_S\}_{K_2}$
4. $A \to C(B) : \{a, K, N_1, t_S\}_{K_2}, \{N_1, t_A\}_K$

(Here N_1, N_2 and N_3 are generated by A, A and C, respectively. K_1, K_2 and K are same as process π_1. t_A and t_S are timestamps generated by timestamp-setting rules for A and S, respectively.)

As for process π_1 we can observe that if and only if the time t' satisfies the following condition

$$t_A + H \leq t' < t_A + D_1 \cdots\cdots (1)$$

$S \models_i^{\pi_1} \sharp(\langle b, N_1, t_A \rangle)$ is valid, and $S \models_i^{\pi_1} A |\overset{*}{\sim} \langle b, N_1, t_A \rangle$ and $S \models_i^{\pi_1} A \ni N_1$ are also valid (cf. a typical BAN-proof of $A \models B |\overset{*}{\sim} s$ and $A \models B \ni s$ can be seen in Section 3.2). This is because process-representation of the position i is as follows and the time-constraint condition $t' < t_A + D_1$ is valid, hence the condition of the evaluation criterion 9 is satisfied.

$$\vdots$$

$$\frac{\Gamma \vdash \Delta, S(u(\langle b, N_1, t_A\rangle, D_1)), Time(t')}{t' < t_A + D_1, \Gamma \vdash \Delta, S(\langle b, N_1, t_A\rangle), Time(t')}\ \text{timestamp}$$

$$\vdots$$

As for process π_2, we can observe that if and only if the time t' satisfies the following condition

$$t_A + 2H \leq t' < t_A + D_1 \cdots\cdots (2)$$

$S \models_i^{\pi_2} \sharp(\langle b, N_1, t_A\rangle)$, $S \models_i^{\pi_2} A \mid\!\overset{*}{\sim} \langle b, N_1, t_A\rangle$ and $S \models_i^{\pi_2} A \ni N_1$ are valid.

Thus, if the negation of the condition (2)

$$t' < t_A + 2H \quad \text{or} \quad t_A + D_1 \leq t' \cdots\cdots (2')$$

is satisfied, then we cannot derive the statements $S \models_i^{\pi_2} \sharp(\langle b, N_1, t_A\rangle)$ nor $S \models_i^{\pi_2} A \mid\!\overset{*}{\sim} \langle b, N_1, t_A\rangle$ nor $S \models_i^{\pi_2} A \ni N_1$.

Therefore from the above results (1) and (2′), as long as t' satisfies the following condition

$$t_A + H < t' < \min\{t_A + D_1, t_A + 2H\}$$

$S \models_i \sharp(\langle b, N_1, t_A\rangle)$, $S \models_i A \mid\!\overset{*}{\sim} \langle b, N_1, t_A\rangle$ and $S \models_i A \ni N_1$ are valid in the standard process (which follows the BAN-Yahalom protocol with respect to $P := B$, $Q := A$ and $R := S$), however these are not valid in the attacked process π_2.

We also consider A's knowledge on freshness in the standard process which follows the BAN-Yahalom protocol with respect to $P := A$, $Q := B$ and $R := S$ (say π_1') as well as in the attacked process π_2. We assume that A checks the freshness of $\langle b, t_S, K, N_1\rangle$ at time t' in π_1' and π_2. In this setting by the same reason of the above example, as long as t' satisfies the following condition

$$t_S + H < t' < \min\{t_S + D_2, t_S + 2H\}$$

$A \models \sharp(\langle b, t_S, K, N_1\rangle)$ is valid in π_1', however not valid in the attacked process π_2.

5 Concluding Remark and Further Works

In this paper we have given a framework of proof-representation of process/protocol and defined some notions about security protocol study (e.g., "a process following a protocol") in this framework. Moreover, we have given an evaluation framework about each participant's knowledge/belief of proof-representation of a process. Especially, we introduce the distinction of belief and knowledge. Based on these frameworks, we have made an evaluation mechanism of freshness more powerful and expressive than the original BAN logic by means of the method of Kanovich-Okada-Scedrov [12]. These frameworks which we have given as a whole are the integration of protocol reasoning methods based on BAN logic within the multiset rewriting model. Up to now the BAN-reasoning framework and the transitional (multiset rewriting) framework have been considered competitors rather than allieds. We have shown, however, that these two can be combined to provide an integrated framework for the security protocol study.

One of further works of this paper is to investigate meaningful sufficient conditions for "if $A \mid\!\equiv \varphi$ then actually φ". (We point out that in our evaluation framework "if $A \models \varphi$ then actually φ" in general.) This is essential for further applications of our framework to the security verification.

Acknowledgement. We would like to thank our US collaborative research teams under the NSF-JSPS joint project. Our research in this paper was carried out based on discussion with and suggestion from our US partner team members; in particular, our proof-representation framework was based on discussion with I.Cervesato-N.A.Durgin-P.Lincoln-J.Mitchell-A.Scedrov's group at the occasions of three US-Japan meetings (in March 2000, Oct. 2001 and Nov. 2002), and on their former work, as well as discussion with/suggestion from M.Abadi. Our framework of time analysis is based on collaborative research with M.Kanovich-A.Scedrov's group. We would like to express our thanks to those US researchers for discussions and suggestions on this paper. We would also like to express our thanks to the anonymous referees for their comments.

References

1. C. Boyd and W. Mao. On a Limitation of BAN Logic. LNCS Vol. 765, Eurocrypt '93, edited by T. Helleseth, pp. 240–247, 1993.
2. M. Burrows, M. Abadi and R. Needham. A Logic of Authentication. *Technical Report 39*, Digital System Research Center, 1989.
3. F. Butler, I. Cervesato, A. Jaggard, and A. Scedrov. A formal analysis of some properties of Kerberos 5 using MSR. In: *S. Schneider, ed., 15-th IEEE Computer Security Foundations Workshop*, Cape Breton, Nova Scotia, Canada, June, 2002, IEEE Computer Society Press, 2002, pp. 175–190.
4. I. Cervesato, N.A. Durgin, P.D. Lincoln, J.C. Mitchell and A. Scedrov. A meta-notation for protocol analysis. *12th IEEE Computer Security Foundations Workshop*, 1999.
5. J. Clark and J. Jacob. A Survey of Authentication Protocol Literature: Version 1.0 (web draft).
6. N.A. Durgin, P.D. Lincoln, J.C. Mitchell and A. Scedrov. Undecidability of bounded security protocol. *The 1999 Federated Logic Conference (FLoC '99)*, 11 pages, 1999.
7. N. Durgin, J. Mitchell, and D. Pavlovic. A Compositional Logic for Protocol Correctness. In: *S. Schneider, ed., 14-th IEEE Computer Security Foundations Workshop*, Cape Breton, Nova Scotia, Canada, June, 2001, IEEE Computer Society Press, 2001.
8. J.-Y. Girard. Linear Logic. *Theoretical Computer Science*, Vol. 50, pp. 1–102, 1987.
9. L. Gong, R. Needham and R. Yahalom. Reasoning About Belief in Cryptographic Protocols. *Proceedings 1990 IEEE Symposium on Research in Security and Privacy*, pp. 234–248, 1990.

10. G. Lowe. An Attack on the Needham-Schroeder Public Key Authentication Protocol. *Information Processing Letters*, 56 (3), pp. 131–136, 1995.
11. R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM* pp. 993–999, 1978.
12. M. Kanovich, M. Okada and A. Scedrov. Specifying Real-Time Finite-State Systems in Linear Logic. *Proc. International Workshop on Time-Sensitive Constraint Programming* and *Electronic Notes of Theoretical Computer Science 16*, No. 1, 14 pages, 1998.
13. M. Okada and K. Hasebe Logical Verifications for Security Protocols Based on Linear Logic. (in Japanese) *Technical report of IEICE*, 102(91), pp. 49–54, 2002.
14. P. Syverson. A taxonomy of replay attacks. *Proceedings of the Computer Security Foundations Workshop (CSFW7)*, pp. 187–191, 1994.
15. P. Syverson and P. C. van Oorschot. A unified cryptographic protocol logic. *NRL Publications*, 5540–227, Naval Research Lab, 1996.
16. P. Syverson and I. Cervesato. The Logic of Authentication Protocols. *Lecture Notes in Computer Science*, Vol. 2171, pp. 63–136, 2001.

Appendix

A The Logical System for Representing Processes-as-Proofs Interpretation

The Language

Sorts and **Terms**: See Section 1 for the definition of sorts and terms.

Predicates: $A(*), B(*), \ldots$ are used for participant predicate constant. $P(*)$, $Q(*), \ldots$ are used for participant predicate variables. ($*$ is any term of `Message`.) $KP(*_1, *_2), Net(*)$. ($*$ is any term of `Message` and $*_1, *_2$ are terms of sorts `PublicKey` and `SecretKey` respectively.)

Formulas: The logical connectives of our language are only $\otimes$ of the linear logic and the first-order quantifiers ($\forall$ and $\exists$).

Axioms and Inference Rules

Logical Inference Rules ($\otimes$ Introduction Rule):

$$\frac{\vdash \Gamma, \varphi \quad \vdash \Delta, \psi}{\vdash \Gamma, \Delta, \varphi \otimes \psi}$$

for any formula φ and ψ.

We define axioms and inference rules about each participant A as follows:

Axioms for A:
nonce generation:

$\vdash \exists n\ A(n)$ (n is bound variable with the sort `Nonce`.)

name generation:
$\vdash \forall p\ A(p)$ (p is bound variable with the sort `Name`.)

key generation:
$\vdash \forall k\ A(k)$ (k is bound variable with the sort `Key`.)

public keys:
$\vdash A(K'), KP(K, K')$

shared keys:
$\vdash A(K), B(K)$

Special Inference Rules for A:

- $\exists-$elimination rule and $\forall-$elimination rule (See Section 2.1 for a remark on $\exists-$elimination rule). We also use the usual left- and right- $\exists-$introduction rules in Section 4.
- Concatenation rules for A

$$\frac{\vdash \Gamma, A(\langle s_1, \ldots, s_n \rangle)}{\vdash \Gamma, A(s_1), \ldots, A(s_n)} \qquad \frac{\vdash \Gamma, A(s_1), \ldots, A(s_n)}{\vdash \Gamma, A(\langle s_1, \ldots, s_n \rangle)}$$

sending:

$$\frac{\vdash \Gamma, A(s)}{\vdash \Gamma, A(s), Net(s)}$$

receiving:

$$\frac{\vdash \Gamma, Net(s)}{\vdash \Gamma, A(s)}$$

encryption (shared key):

$$\frac{\vdash \Gamma, A(K), A(s)}{\vdash \Gamma, A(K), A(s), A(\{s\}_K)}$$

(for any K:`SharedKey`)

decryption (shared key):

$$\frac{\vdash \Gamma, A(K), A(\{s\}_K)}{\vdash \Gamma, A(K), A(\{s\}_K), A(s)}$$

(for any K:`SharedKey`)

encryption (public key):

$$\frac{\vdash \Gamma, KP(K, K'), A(s)}{\vdash \Gamma, KP(K, K'), A(s), A(\{s\}_K)}$$

(for any K:`PublicKey` and for any K':`SecretKey`)

decryption (public key):

$$\frac{\vdash \Gamma, KP(K, K'), A(K'), A(\{s\}_K)}{\vdash \Gamma, KP(K, K'), A(K'), A(\{s\}_K)), A(s)}$$

(for any K:`PublicKey` and for any K':`SecretKey`)

encryption (secret key):

$$\frac{\vdash \Gamma, A(K), A(s)}{\vdash \Gamma, A(K), A(s), A(\{s\}_K)}$$

(for any K: `SecretKey`)

decryption (secret key):

$$\frac{\vdash \Gamma, KP(K, K'), A(\{s\}_{K'})}{\vdash \Gamma, KP(K, K'), A(\{s\}_{K'}), A(s)}$$

(for any K:`PublicKey` and for any K':`SecretKey`)

B The Modified BAN Logic for Evaluation of Belief

Notations: The letters $A, B, C, \ldots$ are used to denote specific participants ($P, Q, R, \ldots$ are used to denote parameterized participants). As for the term level objects, we use the same symbols as syntactic level.

Statements: Atomic statements are as follows: $P \overset{K}{\leftrightarrow} Q$ (K is a shared key for P and Q), $\overset{K}{\mapsto} P$ (P has K as a public key), $P \overset{s}{\rightleftharpoons} Q$ (s is a shared secret with P and Q), $P \triangleleft s$ (P has received s), $P \ni s$ (P has s), $P \mid\!\sim s$ (P said s), $P \mid\!\overset{*}{\sim} s$ (Q says s), $\sharp(s)$ (s is fresh), $Gen(P, s)$ (s is generated by P). When φ is atomic statement, then $P \mid\!\equiv \varphi$ and $P \models \varphi$ are statements.

Inference Rules for Evaluation:

Freshness:

$$\frac{A \mid\!\equiv_i \sharp(s) \quad A \mid\!\equiv_i B \mid\!\sim s}{A \mid\!\equiv_i B \mid\!\overset{*}{\sim} s} \qquad \frac{A \mid\!\equiv_i \sharp(s)}{A \mid\!\equiv_i \sharp(\langle s, u\rangle)} \qquad \frac{A \mid\!\equiv_i \sharp(u)}{A \mid\!\equiv_i \sharp(\langle s, u\rangle)} \qquad \frac{A \mid\!\equiv_i Gen(A, N)}{A \mid\!\equiv_i \sharp(N)}$$

Possession:

$$\frac{A \mid\!\equiv_i B \mid\!\overset{*}{\sim} s}{A \mid\!\equiv_i B \ni s} \qquad \frac{A \mid\!\equiv_i B \triangleleft s}{A \mid\!\equiv_i B \ni s}$$

Concatenation:

$$\frac{A \mid\!\equiv_i B \ni \langle s_1, s_2, \ldots, s_n\rangle}{A \mid\!\equiv_i B \ni s_k} \qquad \frac{A \mid\!\equiv_i B \triangleleft \langle s_1, s_2, \ldots, s_n\rangle}{A \mid\!\equiv_i B \triangleleft s_k} \qquad \frac{A \mid\!\equiv_i B \mid\!\overset{*}{\sim} \langle s_1, s_2, \ldots, s_n\rangle}{A \mid\!\equiv_i B \mid\!\overset{*}{\sim} s_k}$$

Decryption:

$$\frac{A \mid\!\equiv_i B \triangleleft \{s\}_K \quad A \mid\!\equiv_i \overset{K}{\mapsto} B}{A \mid\!\equiv_i B \triangleleft s} \qquad \frac{A \mid\!\equiv_i B \triangleleft \{s\}_K \quad A \mid\!\equiv_i B \ni s}{A \mid\!\equiv_i B \triangleleft s}$$

$$\frac{A \mid\!\equiv_i B \triangleleft \{s\}'_K \quad A \mid\!\equiv_i B \ni K \quad A \mid\!\equiv_i KP(K, K')}{A \mid\!\equiv_i B \triangleleft s}$$

On BAN Logics for Industrial Security Protocols

Nesria Agray[1], Wiebe van der Hoek[1,2], and Erik de Vink[3,4]

[1] Institute of Information and Computing Sciences, Utrecht University, The Netherlands N.Agray@fss.uu.nl, wiebe@cs.uu.nl
[2] Department of Computer Science, University of Liverpool, United Kingdom
[3] Department of Computer Science, Technische Universiteit Eindhoven, The Netherlands evink@win.tue.nl
[4] Leiden Institute of Advanced Computer Science, Leiden University, The Netherlands

Abstract. This paper reports on two case-studies of applying BAN logic to industrial strength security protocols. These studies demonstrate the flexibility of the BAN language, as it caters for the addition of appropriate constructs and rules. We argue that, although a semantical foundation of the formalism is lacking, BAN logic provides an intuitive and powerful technique for security analysis.

1 Introduction

New and upcoming techniques for the internet and also wireless telecommunication require or encourage agents to interchange more and more sensitive data, like payment instructions in e-commerce, strategic information between commercial partners, or personal information in for instance medical applications. Issues like authentication of the partners in a protocol, and the confidentiality of information therefore become of increasing importance: cryptographic protocols are used to distribute keys and authenticate agents and data over hostile networks. Although many of the protocols used look very intricate and hence waterproof, many examples are known of sensitive applications that were cracked and then furnished with new, 'improved' protocols.

The application of logical tools to the analysis of security protocols was pioneered by Burrows, Abadi and Needham. In [4] and [9] specific epistemic logics, collectively referred to as BAN logic, are proposed to deal with authentication issues. Earlier, standard epistemic logic ([6,12]) has been applied successfully to reason about communication protocols, cf. the derivation of the alternating bit protocol in [11] or, more recently, the analysis of TCP [14]). In this setting, contrary to the security set-up of BAN, the implicit assumption is always that the network is not hostile. The work on epistemic logic has the last decades mainly focused on theoretical issues like variants of modal logic (cf. also [7] where epistemic notions are mixed with deontic ones), completeness, and derived notions like Distributed Knowledge and Common Knowledge.

In this paper we discuss the application of the BAN-framework to two of the most widely used security and authentication protocols. The first is applied in the Global System for Mobile communication (GSM, see [10]), an integration of several national cellular networks, which is now world-wide in use to transmit voice-data on the air, but also to send and receive e-mail, faxes and to get access to the internet. The second protocol, Secure Electronic Transactions (SET, see [13]) is an electronic commerce security protocol for doing business over the internet. It is developed under auspices of Visa and Mastercard, and well accepted to support credit card payments over the internet.

The aim of our exercise to apply BAN to those two protocols is the following. First, it goes without saying that such widely used but sensitive protocols have at least partially to be formally validated. Since the protocols under consideration are apparently not derived from a formal specification, before verifying any security goals on them one has to first trim down the massive manuals about the protocols to their bare essentials. Second, we want to demonstrate the flexibility of the BAN-framework by showing how specific issues of the two protocols can be smoothly accounted for by suitable extensions of the BAN-language and reasoning rules Here, we give an indication of the needed extensions for each of the protocols; full proofs are provided in [2]. Finally, the main research question behind our project is in which direction further fundamental work on BAN, or any logic for security and authentication, should proceed to sensibly deal with protocols that are in development or applied on a scale like GSM and SET.

2 BAN Logic

BAN logic is a modal logic with primitives which describe the beliefs of agents involved in a cryptographic protocol. Using the inference rules of BAN logic the evolution of the beliefs of agents during a cryptographic protocol can be studied. Here we present some typical rules that we put to work, on the fly explaining the language of BAN.

The BAN-formalism is built on three sorts of objects: the agents involved in a security protocol, the encryption/decryption and signing/verification keys that the agents possess, and the messages exchanged between agents. The notation $\{M\}_K$ denotes a message encrypted using a key K. For a symmetric key K we have $\{\{M\}_K\}_K = M$ for any message M i.e., decrypting with key K a message M that is encrypted with K reveals the contents M. For a key pair $\langle EK, DK \rangle$ of a public encryption key EK and a private decryption key DK it holds that $\{\{M\}_{EK}\}_{DK} = M$ for any message M. Likewise, for a key pair $\langle SK, VK \rangle$ of a private signing key SK and a public verification key VK it holds that $\{\{H\}_{SK}\}_{VK} = H$ for any hash value H. Hash values are obtained by application of a one-way collision-free hash-function. Proving the hash value $H(m)$ of a message m is a means to demonstrate that m is known, without revealing it (cf. also [2]).

In BAN, we have a number of operators describing the beliefs of agents, for which the usual modal properties apply, like P `believes` $(A \rightarrow B) \rightarrow (P$ `believes` $A \rightarrow P$ `believes` $B)$. On top of that we have the operators sees and possesses (cf. [4,9]). The following rules are an illustration of some of the authentication and encryption rules:

(1) P believes secret$(K, P, Q) \wedge P$ sees $\{X\}_K$
$\quad \rightarrow P$ believes Q said X

(2) P believes belongs_to $(VK, Q) \wedge P$ sees $\{X\}_{SK}$
$\quad \rightarrow P$ believes Q said X

(3) P believes fresh$(X) \wedge P$ believes Q said X
$\quad \rightarrow P$ believes Q believes X

(4) P possesses $DK \wedge P$ sees $\{X\}_{EK} \rightarrow P$ sees X

Intuitively, **(1)** says that if an agent P believes that it shares the symmetric key K with an agent Q, and agent P receives a message X encrypted under K, then agent P believes that agent Q once said message X. This rule addresses symmetric encryption. In a similar way, **(2)** models digital signatures. If an agent P believes that the verification key VK belongs to an agent Q, then P concludes, confronted with a message or hash encrypted with the corresponding signing key SK, that the message or hash originates from the agent Q. Regarding rule **(3)**, if an agent P believes that certain information is new, i.e. constructed during the current protocol run, and P furthermore believes that Q conveyed this information, then P concludes that the agent Q believes himself this information. (Underlying this is the overall assumption in BAN logic of honesty of agents; the participating agents' behavior is consistent with the particular protocol.) According to **(4)**, if an agent P sees an encrypted message and P possesses the decryption key then P can read the message itself.

Analysis of an authentication protocol using BAN-logic is comprised of four steps: (1) first, the initial beliefs of the participating agents are formulated; (2) the protocol security goals are stated; (3) the effect of the protocol's messages is formalized in BAN and (4), the final beliefs are shown to fulfil the goals.

3 Analyzing the GSM Security Protocol

The GSM protocol, Global System for Mobile communication, is a protocol for mobile telephony operating world-wide since the beginning of the 1990s. Abstracting away, in our analysis, from a lot of details we distinguish three parties involved in the protocol: the mobile station MS, which is the subscriber's interface to the network, connected via a radio link, the home location register HLR which is an intelligent database where a subscriber's specific parameters are stored, and the visitor location register VLR, a database that temporarily stores user specific parameters. The VLR's role is to support the HLR by handling most of the user-related queries.

If a mobile station MS wants the GSM service, it must prove its identity to the network. To this end, the mobile station sends a service request via the radio path to the VLR. The VLR then forwards this identity to the HLR, which generates a fresh random number RAND and computes a result RES by applying a one-way and collision free function $\mathtt{A}_3$ to a key $\mathtt{K}$ and $\mathtt{RAND}$, i.e. $\mathtt{RES} = \mathtt{A}_3(\mathtt{K}, \mathtt{RAND})$. The identity key $\mathtt{K}$ is a shared secret between the $\mathtt{HLR}$ and the $\mathtt{MS}$, but not known to $\mathtt{VLR}$. If $\mathtt{MS}$ is able to generate the same value $\mathtt{RES}$ when provided with $\mathtt{RAND}$, it has proven its identity. The $\mathtt{HLR}$ also computes an encryption key $\mathtt{K}'$ that will ensure privacy of $\mathtt{MS}$ when it further communicates with the $\mathtt{VLR}$. This is done using an algorithm $\mathtt{A}_8$, hence $\mathtt{K}' = \mathtt{A}_8(\mathtt{K}, \mathtt{RAND})$. The $\mathtt{HLR}$ then sends the triple $\langle \mathtt{RAND}, \mathtt{RES}, \mathtt{K}' \rangle$ to the $\mathtt{VLR}$. Upon receipt of this triple, the $\mathtt{VLR}$ forwards $\mathtt{RAND}$ to the $\mathtt{MS}$ and waits for an appropriate response from the $\mathtt{MS}$. The mobile station can calculate $\mathtt{RES}$ from the key $\mathtt{K}$ and the challenge $\mathtt{RAND}$ using the algorithm $\mathtt{A}_3$. Then, $\mathtt{MS}$ uses $\mathtt{RAND}$ together with its key K and the algorithm $\mathtt{A}_8$ to calculate the session key $\mathtt{K}'$ and stores it locally.

We next give the crucial rules that we add to BAN. For the modeling we add various auxiliary functions and predicates like `ch` and `comp`. Free variables are assumed to be universally quantified.

(4) $\mathtt{VLR\ believes}\ [\mathtt{HLR\ said\ ch}(x,y,z) \rightarrow \mathtt{rand}(x) \wedge \mathtt{res}(y) \wedge \mathtt{skey}(z)]$

(5) $\mathtt{VLR\ believes}\ [(\mathtt{MS\ said\ res}(y) \wedge \mathtt{HLR\ said\ ch}(x,y,z) \wedge \mathtt{fresh}(x))$
$\quad \rightarrow \mathtt{MS\ possesses}\ z]$

(6) $P\ \mathtt{believes}\ (P\ \mathtt{possesses}\ K \wedge Q\ \mathtt{possesses}\ K)$
$\quad \rightarrow P\ \mathtt{believes\ secret}(K,P,Q)$

(7) $\mathtt{MS\ believes}\ [(\mathtt{VLR\ said\ rand}(x)) \wedge \mathtt{fresh}(x) \wedge$
$\quad \mathtt{ikey}(K) \wedge \mathtt{a}_8(A) \wedge K' = \mathtt{comp}(A,x,K) \wedge$
$\quad\quad \mathtt{secret}(\langle K,A \rangle, \mathtt{MS}, \mathtt{HLR}) \rightarrow \mathtt{VLR\ possesses}\ K']$

Rule **(4)** says that the $\mathtt{VLR}$ knows the meaning of the three values in the challenge triplet $\mathtt{ch}(x,y,z)$ received from the $\mathtt{HLR}$. Our rule **(5)** expresses that the session key is calculated by the mobile station at the same time the calculation of the `res`-value takes place. So at the moment that the right fresh `res`-value is received from the mobile station, the $\mathtt{VLR}$ knows that the mobile station is in possession of the right session key. Rule **(6)** is an obvious, but tricky rule. If some agent R possesses the key K as well, it is no longer justified to consider K a secret between P and Q. Finally, concerning rule **(7)**, when the mobile station receives a random challenge, it computes the corresponding response value and session key using the appropriate secret A8-algorithm, the random value and the secret identity key K that it shares with the $\mathtt{HLR}$. It is implicit in the GSM authentication protocol that the mobile station trusts the network on the matter of the random value, i.e. $\mathtt{MS}$ assumes that it originates from the $\mathtt{HLR}$. Furthermore, it is assumed that at the moment that the $\mathtt{MS}$ receives a random value from the $\mathtt{VLR}$, the $\mathtt{MS}$ concludes that the $\mathtt{VLR}$ possesses the corresponding session key.

We have omitted rules concerning freshness here. Also, space does not allow us to formulate the initial beliefs of the three partners. Instead, we formulate the authentication question and the question concerning the agreement on the encryption key. The $\mathtt{VLR}$ must check whether a mobile station with a given identifier is really registered in the system, and whether the mobile station has the identity it claims. This is done by comparing the `res`-values received from the mobile station at one side and from the HLR at the other side. If the two values match then the VLR knows that it is indeed talking to the mobile station. The first authentication question is as follows:

(Q1) $\exists x : \texttt{fresh}(x) \wedge \texttt{VLR believes res}(x) \wedge \texttt{MS said res}(x)$

The next question that we discuss concerns the agreement on the session key. At the end of the protocol the VLR must know that it shares the right session key with the mobile station. The mobile station must believe the same statement. This is modeled as follows:

(Q2) $\exists z : \texttt{MS believes } (\texttt{fresh}(z) \wedge \texttt{secret}(z, \texttt{VLR}, \texttt{MS})) \wedge$
$\texttt{VLR believes } (\texttt{fresh}(z) \wedge \texttt{secret}(z, \texttt{VLR}, \texttt{MS}))$

In our full paper [2], we specify a set of initial beliefs of the three parties involved, translate the effect of the messages into BAN, and prove, amongst others, **(Q1)** and **(Q2)**.

4 Analyzing the SET Protocol

In this section we describe the SET-protocol (see [13]). SET stands for *Secure Electronic Transactions* and is designed for credit card payments over the Internet. By combining both symmetric and public-key cryptographic techniques SET provides several security services at an adequate performance. In particular, SET provides message integrity together with blinding of information, such that (i) no banking information of the customer is revealed to the merchant, and (ii) no purchase information of the customer is disclosed to the bank. For this latter services SET deploys so-called dual signatures.

Let us first explain standard digital signatures. Each agent participating in a run of SET holds a pair of keys $\langle SK, VK\rangle$ consisting of a *signing key* and a *verification key*. The signing key is private, the verification key is public. If Alice wants to sign a message M, she first computes the hash value $\texttt{hash}(M)$ of M, encrypts this value with her signing key SK and attaches this signature $\{\texttt{hash}(M)\}_{SK}$ to the message M, thus yielding $(M, \{\texttt{hash}(M)\}_{SK})$. When Bob receives a signed message (M, S), presumably coming from Alice, he first computes the hash value $\texttt{hash}(M)$ of M, then he retrieves the corresponding verification key VK of Alice, decrypts the signature S with the key VK, and checks whether the resulting value $\{S\}_{VK}$ equals the earlier value $\texttt{hash}(M)$. Note, since $\langle SK, VK\rangle$ is a key pair with private key SK and public key VK, it holds that $\{\{X\}_{SK}\}_{VK} = X$. So, if the two values agree, Bob concludes that Alice signed the message as she is the only agent in possession of the signing key that is needed to produce the signature for the hash value of the message. The BAN rule that we have used to model this is

(8) $P \texttt{ sees } (M, S) \wedge P \texttt{ believes } VK \texttt{ belongs\_to } Q \wedge$
$\texttt{hash}(M) = \{S\}_{VK} \rightarrow P \texttt{ believes } Q \texttt{ said } M.$

Next we discuss dual signatures. SET makes use of so-called dual signatures to blind some information for one agent while showing the same information to another one, but in the mean time providing the two agents with a reference to the information. Sending each part of the information separately to the intended parties does not work, because the two parts of information are related and this relationship is important for the remainder of the protocol. For example, a purchase request from a card holder C contains both order data and payment data. The payment data is intended for the bank B and must be hidden from the merchant M. Dually, it is no business of the bank to get to know the actual goods that the customer buys. However, at some point in time after the particular purchase request, the merchant M want to capture his money for the purchase from the bank B. For this M and B need to refer to the same procurement.

To see this at work, suppose that an agent P has signing key SK, verification key VK and wants to send some information I to an agent Q. Additionally, Q needs a way to link I to some other information J that will not be revealed to him. To realize this, the agent P sends Q the message

$$(I, \mathtt{hash}(J), \{\mathtt{hash}(\mathtt{hash}(I) \mid \mathtt{hash}(J))\}_{SK}),$$

where the so-called dual hash $\mathtt{hash}(\mathtt{hash}(I) \mid \mathtt{hash}(J))$ is obtained by hashing the juxtaposition of $\mathtt{hash}(I)$ and $\mathtt{hash}(J)$. By encryption of the dual hash with her private signing key, agent P testifies that both the information I and the hash $\mathtt{hash}(J)$ originate from her. The agent Q can verify this by using the corresponding public verification key of P, similar to the case of standard digital signatures as described above. However, Q does not know anything about the information J. Now, if yet some other agent R obtains from P the message

$$(\mathtt{hash}(I), J, \{\mathtt{hash}(\mathtt{hash}(I) \mid \mathtt{hash}(J))\}_{SK}),$$

she knows that the message indeed has been sent by P, but has no clue about the information I. The crucial point is that agents Q and R both are aware that $I\&\mathtt{hash}(J)$ and $\mathtt{hash}(I)\&J$, respectively, are related. Our relevant BAN rule is as follows:

(9) P $\mathtt{sees}$ $(I, G, \{H\}_{SK}) \wedge P$ $\mathtt{believes}$ $\mathtt{belongs\_to}$ $(VK, Q) \wedge$
$\mathtt{hash}(\mathtt{hash}(I)|G) = H \rightarrow P$ $\mathtt{believes}$ $(Q$ $\mathtt{said}$ $I \wedge \mathtt{dhash}_1(I, G, H))$

Here, $\mathtt{dhash}_1$ is a predicate stating that the hash of I concatenated with G will hash to H. There is also a similar rule for a message $(F, J, \{H\}_{SK})$ for a hash value F, a message J and dual hash H using an auxiliary predicate $\mathtt{dhash}_2$.

Besides user authentication, during a run of SET, the card holder, the merchant and the payment gateway have to agree on a purchase and the corresponding cost. So after the run of the protocol, the three participants must store consistent information about the purchase. For example, the three parties must store the same purchase amount. This is achieved when the three parties agree on the linkage between the order data and the payment instructions. As example, for P, this is modeled in security requirement **(Q3)** as follows:

(Q3) P $\mathtt{believes}$ $\big(\mathtt{pay\_instr}(C, \mathtt{PData}) \wedge (\exists \mathtt{OData} : \mathtt{link}_2(\mathtt{OData}, \mathtt{PData}) \wedge$
M $\mathtt{believes}$ $\mathtt{link}_1(\mathtt{OData}, \mathtt{PData}) \wedge$
C $\mathtt{believes}$ $\mathtt{link}_1(\mathtt{OData}, \mathtt{PData}) \wedge \mathtt{link}_2(\mathtt{OData}, \mathtt{PData}))\big)$

The payment gateway P cannot see the order data, but believes in the existence of an order $\mathtt{OData}$ generated by the card holder which is linked to the card holder's payment instructions. Therefore, the well-known Barcan-formula P `believes` $\exists x : \phi \rightarrow \exists x : P$ `believes` ϕ is not valid in our framework. Furthermore, P has to trust this link, and has to know that both the merchant and the card holder believe in the same link. Similar rules apply to M and C.

In [2] we present the translation of the protocol in BAN. In order to give the reader a flavor of this, let us consider the third transaction of the protocol, in which the card holder sends its purchase request to the merchant.

(PReq) $C \rightarrow M$:
$(\texttt{OData}, \texttt{hash}(\texttt{PData}), \{\texttt{hash}(\texttt{hash}(\texttt{OData}) \mid \texttt{hash}(\texttt{PData}))\}_{SKC}),$
$\{\texttt{PData}, \texttt{hash}(\texttt{OData}), \{\texttt{hash}(\texttt{hash}(\texttt{OData}) \mid \texttt{hash}(\texttt{PData}))\}_{SKC}\}_{K},$
$\{K\}_{EKP},$
$\texttt{sigCert}(C, SKC))$

That is, card holder C sends to merchant M the following: (1) a message containing the order data and the hash of the payment data as clear text, but dually signed with the signing key of agent C; (2) a message containing the hash value of the order data and the payment data, dually signed with the signing key of C and encrypted with a random session key K; (3) the session key K encrypted by the payment gateway's encryption key EKP, so that only the payment gateway, hence not the merchant, can learn this session key and, consequently, the payment data encrypted with the key K; (4) a public-key certificate $\texttt{sigCert}(C, SKC)$ for the cardholder's signing key which includes the corresponding verification key.

In order to compute the effects of this message on the beliefs of M, one first has to know the beliefs of M prior to performing the transaction **(PReq)**. For example, one belief concerns the possession of the verification key of the root Certification Authority for the public-key certificate for C's signing key. It is clear that the analysis makes such an assumption explicit. Although the SET protocol is rather detailed, in [2] we were able to give a formal proof of **(Q3)** as mentioned, as well as for other authentication goals.

5 Future Research

We see the following lines of research triggered by the experiences reported in this paper. First, the formidable proofs in [2] call for tool-support for the verification of non-trivial security goals. Apart from automated translation, our experiences reveal the need for flexible computer-assistance to incorporate logical extensions and to guide proof construction in the adapted framework.

Second, it is well known that the semantics of BAN logic is under debate (cf. [1,15,16,3]). At present, we cannot claim that our rules are sound or complete. On the one hand, this questions the impact of the derived results (what does it mean that some string has been derived?), but on the other hand, strengthens the call for an adequate model. To our opinion the various proposed semantics for BAN hardly allow for changes in the logic. This raises the question for a model which easily treats extensions as described in this paper. In the protocols considered not only information is exchanged, but also requests. Therefore, one might want to stress the goals of an agent as part of his mental state. However, the framework should not only reflect the state of the agent, but also the *dynamics*: agents learn from the messages they receive, thus calling for a language that deals with knowledge together with epistemic actions like learning and updating. To this aim we try in a current project to exploit the knowledge games semantics of [5] in the setting of authentication logics.

References

1. M. Abadi and M. Tuttle, A Semantics for a Logic of Authentication, in *Proceedings of the ACM Symposium on Principles of Distributed Computing,* p. 201–216, 1991.
2. N. Agray, *The BAN Approach to Formal Verification: Authentication in GSM and SET,* Master Thesis, Utrecht University Number INF/SCR-01-09 and KPN Research, Leidschendam (2001).
3. A. Bleeker and L. Meertens, A Semantics for BAN Logic, *Proc. DIMACS workshop on Design and Formal Verification of Protocols,* 1997, `http://dimacs.rutgers.edu/Workshops/Security/program2/program.html`.
4. M. Burrows, M. Abadi and R. Needham, A Logic of Authentication, *ACM Transactions on Computer Systems,* vol. 8, p. 18–36, 1990.
5. H. van Ditmarsch, *Knowledge Games.* PhD. thesis, Groningen, 2000, available at `http://tcw2.ppsw.rug.nl/~hans`.
6. R. Fagin, J.Y. Halpern, Y. Moses and M.Y. Vardi, *Reasoning About Knowledge,* MIT Press, 1995.
7. M. Fasli, On Commitments, Roles and Obligations. This volume.
8. M. Fitting and R. Mendelsohn, *First order modal logic,* Kluwer, 1998.
9. L. Gong, R. Needham and R. Yahalom, Reasoning about Belief in Cryptographic Protocol Analysis, *Proc. IEEE Symp. on Research in Security and Privacy,* p. 234–248, 1990.
10. H. Gunnar, *GSM Networks: Protocols, Terminology, and Implementations,* Artech House 1999.
11. J.Y. Halpern and L.D. Zuck, A Little Knowledge Goes a Long Way: Simple Knowledge-Based Derivations and Correctness Proofs for a Family of Protocols, *Proc. 6th ACM Symp. on Principles of Distributed Computing,* 1987, p. 268–280.
12. J.-J.Ch. Meyer and W. van der Hoek, *Epistemic Logic for AI and Computer Science,* Cambridge University Press, 1995.
13. The SET Standard Book 2: Programmer's Guide, version 1.0, SETCO 1997, `www.setco.org`.
14. F. Stulp and R. Verbrugge, A knowledge-based algorithm for the Internet protocol TCP, to appear in the *Bulletin of Economic Research,* 2001. Also at `http://tcw2.ppsw.rug.nl/prepublications`
15. P. Syverson, The Use of Logic in the Analysis of Cryptographic Protocols, in *Proc. IEEE Symp. on Research in Security and Privacy,* 1991.
16. G. Wedel and V. Kessler, Formal Semantics for Authentication Logics, *Proc. ESORICS'96,* p. 219–241, 1996.

TEMPORAL AND MODAL LOGIC[*]

by

E. Allen Emerson
Computer Sciences Department
University of Texas at Austin
Austin, Texas 78712
USA

March 14, 1995

Abstract

We give a comprehensive and unifying survey of the theoretical aspects of Temporal and Modal Logic. (Note: This paper is to appear in the *Handbook of Theoretical Computer Science*, J. van Leeuwen, managing editor, North-Holland Pub. Co.)

[*]Work supported in part by US NSF Grant CCR8511354 , ONR Contract N00014-86-K-0763 and Netherlands ZWO grant nf-3/nfb 62-500.

1 Introduction

The class of Modal Logics was originally developed by philosophers to study different "modes" of truth. For example, the assertion P may be false in the present world, and yet the assertion *possibly* P true, if there exists an alternate world where P is true. Temporal Logic is a special type of Modal Logic; it provides a formal system for qualitatively describing and reasoning about how the truth values of assertions change over time. In a system of Temporal Logic, various temporal operators or "modalities" are provided to describe and reason about how the truth values of assertions vary with time. Typical temporal operators include *sometimes* P which is true now if there is a future moment at which P becomes true and *always* Q which is true now if Q is true at all future moments.

In a landmark paper [Pn77] Pnueli argued that Temporal Logic could be a useful formalism for specifying and verifying correctness of computer programs, one that is especially appropriate for reasoning about nonterminating or continuously operating concurrent programs such as operating systems and network communication protocols. In an ordinary sequential program, e.g. a program to sort a list of numbers, program correctness can be formulated in terms of a Precondition/Postcondition pair in a formalism such as *Hoare's Logic* because the program's underlying semantics can be viewed as given by a *transformation* from an initial state to a final state. However, for a continuously operating, *reactive* program such as an operating system, its normal behavior is a nonterminating computation which maintains an ongoing interaction with the environment. Since there is no final state, formalisms such as Hoare's logic which are based on a transformational semantics, are of little use for such nonterminating programs. The operators of temporal logic such as *sometimes* and *always* appear quite appropriate for for describing the time-varying behavior of such programs.

These ideas were subsequently explored and extended by a number of researchers. Now Temporal Logic is an active area of research interest. It has been used or proposed for use in virtually all aspects of concurrent program design, including specification, verification, manual program composition (development), and mechanical program synthesis. In order to support these applications a great deal mathematical machinery connected with Temporal Logic has been developed. In this survey we focus on this machinery, which is most relevant to Theoretical Computer Science. Some attention is given, however, to motivating applications.

The remainder of this paper is organized as follows: In section 2 we describe a multi-axis classification of systems of Temporal Logic, in order to give the reader a feel for the large variety of systems possible. Our presentation centers around only a few—those most thoroughly investigated—types of Temporal Logics. In section 3 we describe the framework of Linear Temporal Logic. In both its propositional and First-order forms, Linear Temporal Logic has been widely employed in the specification and verification of programs. In section 4 we describe the competing framework of Branching Temporal Logic which has also seen wide use. In section 5 we describe how Temporal Logic structures can be used to model concurrent programs using nondeterminism and fairness. Technical machinery for Temporal reasoning is discussed in section 6, including decision procedures and axiom systems. Applications of Temporal Logic are discussed in section 7, while in the concluding section 8 other modal and temporal logics in computer science are briefly described.

2 Classification of Temporal Logics

We can classify most systems of TL (Temporal Logic) used for reasoning about concurrent programs along a number of axes: propositional versus first-order, global versus compositional, branching versus linear, points versus intervals, and past versus future tense. Most research to date has concentrated on global, point-based, discrete time, future tense logics; therefore our survey will focus on representative systems of this type. However, to give the reader an idea of the wide range of possibilities in formulating a system of Temporal Logic, we describe the various alternatives in more detail below.

2.1 Propositional versus First-order

In a propositional TL, the non-temporal (i.e., non-modal) portion of the logic is just classical propositional logic. Thus formulae are built up from *atomic propositions*, which intuitively express atomic facts about the underlying state of the concurrent system, *truth-functional connectives*, such as $\wedge$, $\vee$, $\neg$ (representing "and," "or," and "not," respectively), and the temporal operators. Propositional TL corresponds to the most abstract level of reasoning, analogous to classical propositional logic.

The atomic propositions of propositional TL are refined into expressions built up from variables, constants, functions, predicates, and quantifiers, to get *First-order TL*. There are several different types of First order TLs. We can distinguish between *uninterpreted* First order TL where we make no assumptions about the special properties of structures considered, and *interpreted* First order TL where a specific structure (or class of structures) is assumed. In a *fully* interpreted First order TL, we have a specific domain (e.g. *integer* or *stack*) for each variable, a specific, concrete function over the domain for each function symbol, and so forth, while in a *partially* interpreted First order TL we might assume a specific domain but, e.g., leave the function symbols uninterpreted. It is also common to distinguish between *local* variables which are assigned, by the semantics, different values in different states and *global* variables which are assigned a single value which holds globally over all states. Finally, we can choose to impose or not impose various syntactic restrictions on the interaction of quantifiers and temporal operators. An unrestricted syntax will allow, e.g., modal operators within the scope of quantifiers. For example, we have instances of *Barcan's Formula*: $\forall$y *always* (P(y)) $\equiv$ *always* ($\forall$y P(y)). Such unrestricted logics tend to be highly undecidable. In contrast we can disallow such quantification over temporal operators to get a restricted first-order TL consisting of essentially propositional TL plus a first-order language for specifying the "atomic" propositions.

2.2 Global versus Compositional

Most systems of TL proposed to date are *endogenous*. In an endogenous TL, all temporal operators are interpreted in a single universe corresponding to a single concurrent program. Such TLs are suitable for *global* reasoning about a complete, concurrent program. In an *exogenous* TL, the syntax of the temporal operators allows expression of correctness properties concerning several different programs (or program fragments) in the same formula. Such logics facilitate *compositional* (or *modular*) program reasoning: We can verify a complete program by specifying and verifying its constituent subprograms, and then combining them into a complete program together with its proof of correctness, using the proofs of the subprograms as lemmas (cf. [BKP84], [Pn84]).

2.3 Branching versus Linear Time

In defining a system of temporal logic, there are two possible views regarding the underlying nature of time. One is that the course of time is linear: At each moment there is only one possible future moment. The other is that time has a branching, tree-like nature: At each moment, time may split into alternate courses representing different possible futures. Depending upon which view is chosen, we classify a system of temporal logic as either a linear time logic in which the semantics of the time structure is linear, or a system of branching time logic based on the semantics corresponding to a branching time structure. The temporal modalities of a temporal logic system usually reflect the character of time assumed in the semantics. Thus, in a logic of linear time, temporal modalities are provided for describing events along a single time line. In contrast, in a logic of branching time, the modalities reflect the branching nature of time by allowing quantification over possible futures. Both approaches have been applied to program reasoning, and it is a matter of debate as to whether branching or linear time is preferable (cf. [La80], [EH86], [Pn85])

2.4 Points versus Intervals

Most temporal logic formalisms developed for program reasoning have been based on temporal operators that are evaluated as true or false of *points* in time. Some formalisms (cf. [SMV83], [Mo83], [HS86]),however, have temporal operators that are evaluated over *intervals* of time, the claim being that use of intervals greatly simplifies the formulation of certain correctness properties.

The following related issue has to do with the underlying structure of time.

2.5 Discrete versus Continuous

In most temporal logics used for program reasoning, time is *discrete* where the present moment corresponds to the program's current state and the next moment corresponds to the program's immediate successor state. Thus the temporal structure corresponding to a program execution, a sequence of states, is the nonnegative integers. However, tense logics interpreted over a *continuous* (or *dense*) time structure such as the reals (or rationals) have been investigated by philosophers. Their application to reasoning about concurrent programs was proposed in [BKP86] to facilitate the formulation of fully abstract semantics. Such continuous time logics may also have applications in so-called real-time programs where strict, quantitative performance requirements are placed on programs.

2.6 Past versus Future

As originally developed by philosophers, temporal modalities were provided for describing the occurrence of events in the past as well as the future. However, in most temporal logics for reasoning about concurrency, only future tense operators are provided. This appears reasonable since, as a rule, program executions have a definite starting time, and it can be shown that, as a consequence, inclusion of past tense operators adds no expressive power. Recently, however, it has been advanced that use of the past tense operators might be useful simply in order to make the formulation of specifications more natural and convenient (cf. [LPZ85]). Moreover, past tense operators appear to play an important role in compositional specification somewhat analogous to that of history variables.

3 The Technical Framework of Linear Temporal Logic

3.1 Timelines

In linear temporal logic the underlying structure of time is a totally ordered set (S,<). In the sequel we will further assume that the underlying structure of time is isomorphic to the natural numbers with their usual ordering (|N,<). Note that under our assumption time,

(i) is discrete,

(ii) has an initial moment with no predecessors, and

(iii) is infinite into the future.

We remark that these properties seem quite appropriate in view of our intended application: reasoning about the behavior of ongoing concurrent programs. Property (i) reflects the fact that modern day computers are discrete, digital devices; property (ii) is appropriate since computation begins at an initial state; and (iii) is appropriate since we develop our formalism for reasoning about ongoing, ideally nonterminating behavior.

Let AP be an underlying set of atomic proposition symbols, which we denote by P, Q, P_1, Q_1, P′, Q′, etc. We can then formalize our notion of a timeline as a *linear time structure* M = (S,x,L) where

S — is a set of *states*,
x : |N → S — is an infinite sequence of states, and
L : S → PowerSet(AP) — is a labelling of each state with
the set of atomic propositions in AP true at the state.

We usually employ the more convenient notation x = (s_0, s_1, s_2, ...) = (x(0), x(1), x(2), ...) to denote the *timeline* x. Alternative terminology permits us to refer to x as a *fullpath*, or *computation sequence*, or *computation*, or simply as a *path*; the latter could cause confusion in rare instances since we intend the *maximal path* x, not just one of its prefixes, whence the term fullpath; but ordinarily no confusion will result. (Other synonyms include execution, execution sequence, trace, history, and run.)

Remark: We could convey the same information associating a truth value to each atomic proposition at each state by defining the labelling L as a mapping AP → PowerSet(S) which assigns to each atomic proposition the set of states at which it is true. Another equivalent alternative is to use a mapping L : S $\times$ AP $\rightarrow$ {*true*, *false*} such that L(s,P) = *true* iff it is intended that P be true at s. Still another alternative is to have L : S $\rightarrow$ (AP $\rightarrow$ {*true*, *false*}) so that L(s) is an interpretation of each proposition symbol at state s. In the future, we will use whichever presentation is most convenient for the purpose at hand, assuming the above equivalences to be obvious.

3.2 Propositional Linear Temporal Logic

In this subsection we will define the formal syntax and semantics of Propositional Linear Temporal Logic (PLTL). The basic temporal operators of this system are Fp ("sometime p"; also read as "eventually p"), Gp ("always p"; also read as "henceforth p"), Xp ("nexttime p"), and p U q ("p until q"). Figure 1 below illustrates their intuitive meanings. The formulae of this system are built up from atomic propositions, the truth-functional connectives ($\wedge$, $\vee$, $\neg$, etc.) and the above-mentioned temporal operators. This system, or some slight variation thereof, is frequently employed in applications of temporal logic to concurrent programming.

3.2.1 Syntax

The set of formulae of Propositional Linear Temporal Logic (PLTL) is the least set of formulae generated by the following rules:

1. Each atomic proposition P is a formula.
2. If p and q are formulae then p $\wedge$ q and $\neg$p are formulae.
3. If p and q are formulae then p U q and Xp are formulae.

The other formulae can then be introduced as abbreviations in the usual way: For the propositional connectives, p $\vee$ q abbreviates $\neg(\neg p \wedge \neg q)$, p $\Rightarrow$ q abbreviates $\neg p \vee q$, and p $\Leftrightarrow$ q abbreviates $(p \Rightarrow q) \wedge (q \Rightarrow p)$. The boolean constant *true* abbreviates p $\vee$ $\neg$p, while *false* abbreviates $\neg$*true*. Then the temporal connective Fp abbreviates (true U p) and Gp abbreviates $\neg$F $\neg$p. It is convenient to also have $\overset{\infty}{\mathrm{F}}$p abbreviate GFp (infinitely often), $\overset{\infty}{\mathrm{G}}$p abbreviate FGp ("almost everywhere"), and (p B q) ("p precedes q") abbreviate $\neg(\neg p \text{ U } q)$.

Remark: The above is an *abstract syntax* where we have suppressed detail regarding parenthesization, binding power of operators, and so forth. In practice, we use the following notational conventions, supplemented by auxiliary parentheses as needed: The connectives of highest binding power are the temporal operators F, G, X, U, B, $\overset{\infty}{\mathrm{F}}$, and $\overset{\infty}{\mathrm{G}}$. The operator $\neg$ is of next highest binding power, followed by $\wedge$, followed by $\vee$, followed by $\Rightarrow$, followed finally by $\Leftrightarrow$ as the operator of least binding power.

Example: $\neg p_1 \text{ U } q_1 \wedge r_1 \vee r_2$ means $(\neg(p_1 \text{ U } q_1)) \wedge r_1) \vee r_2$.

3.2.2 Semantics

We define the semantics of a formula p of PLTL with respect to a linear time structure M = (S,x,L) as above. We write M,x $\models$ p to mean that "in structure M formula p is true of timeline x." When M is understood we write x $\models$ p. The notational convention that x^i = the suffix path $s_i, s_{i+1}, s_{i+2}\ldots$ is used. We define $\models$ inductively on the structure of the formulae:

1. $x \models P$ iff $P \in L(s_0)$, for atomic proposition P
2. $x \models p \wedge q$ iff $x \models p$ and $x \models q$
   $x \models \neg p$ iff it is not the case that $x \models p$
3. $x \models (p\ U\ q)$ iff $\exists j\ (x^j \models q$ and $\forall k < j(x^k \models p))$
   $x \models Xp$ iff $x^1 \models p$

The modality (p U q), read as "p until q" asserts that q does eventually hold and that p will hold everywhere prior to q.

The modality Xp, read as "next time p" holds now iff p holds at the next moment.

For conciseness, we took the temporal operator U and X as primitive, and defined the others as abbreviations. However, the other operators are themselves of sufficient independent importance that we also give their formal definitions explicitly.

The modality Fq, read as "sometimes q" or "eventually q" and meaning that at some future moment q is true, is formally defined so that

$x \models Fq$ iff $\exists j\ (x^j \models q)$

The modality Gq, read as "always q" or "henceforth q" and meaning that at all future moments q is true, can be formally defined as

$x \models Gq$ iff $\forall j\ (x^j \models q)$

The modality (p B q), read as "p precedes q" or "p before q" and which intuitively means that "if q ever happens in the future, it is strictly preceded by an occurrence of p," has the following formal definition

$x \models (p\ B\ q)$ iff $\forall j\ (x^j \models q$ implies $\exists k < j(x^k \models p))$

The modality $\overset{\infty}{F}p$, which is read as "infinitely often p," intuitively means that it is always true that p eventually holds, or in other words that p is true infinitely often, can be defined formally as

$x \models \overset{\infty}{F}p$ iff $\forall k\ \exists j \geq k\ x^j \models p$

The modality $\overset{\infty}{G}p$, which is read as "almost everywhere p" or "almost always p," intuitively means that p holds at all but a finite number of times, can be defined as

$x \models \overset{\infty}{G}p$ iff $\exists k\ \forall j > k\ x^j \models p$

3.2.3 Basic Definitions

We say that PLTL formula p is *satisfiable* iff there exists a linear time structure M = (S,x,L) such that x $\models$ p. We say that any such structure defines a *model* of p. We say that p is *valid*, and write $\models$ p, iff for all linear time structures M = (S,x,L) we have x $\models$ p. Note that p is valid iff ¬p is not satisfiable.

3.2.4 Examples

We have the following examples:

p ⇒ Fq intuitively means that "if p is true now then at some future moment q will be true." This formula is satisfiable, but not valid.

G(p ⇒ Fq) intuitively means that "whenever p is true, q will be true at some subsequent moment." This formula is also satisfiable, but not valid.

G(p ⇒ Fq) ⇒ (p ⇒ Fq) is a valid formula, but its converse only satisfiable.

p ∧ G(p ⇒ Xp) ⇒ Gp means that if p is true now and whenever p is true it is also true at the next moment, then p is always true. This formula is valid, and is a temporal formulation of mathematical induction.

(p U q) ∧ ((¬p) B q) means that p will be true until q eventually holds, and that the first occurrence of q will be preceded by ¬p. This formula is unsatisfiable.

Significant Validities

The duality between the linear temporal operators are illustrated by the following assertions:

$\models \mathrm{G}\,\neg p \equiv \neg \mathrm{F}p$
$\models \mathrm{F}\,\neg p \equiv \neg \mathrm{G}p$
$\models \mathrm{X}\,\neg p \equiv \neg \mathrm{X}p$
$\models \overset{\infty}{\mathrm{F}}\neg p \equiv \neg \overset{\infty}{\mathrm{G}}p$
$\models \overset{\infty}{\mathrm{G}}\neg p \equiv \neg \overset{\infty}{\mathrm{F}}p$
$\models ((\neg p)\ \mathrm{U}\ q) \equiv \neg(p\ \mathrm{B}\ q)$

The following are some important implications between the temporal operators, which cannot be strengthened to equivalences:

$\models p \Rightarrow \mathrm{F}p$
$\models \mathrm{G}p \Rightarrow p$
$\models \mathrm{X}p \Rightarrow \mathrm{F}p$
$\models \mathrm{G}p \Rightarrow \mathrm{X}p$
$\models \mathrm{G}p \Rightarrow \mathrm{F}p$
$\models \mathrm{G}p \Rightarrow \mathrm{XG}p$
$\models p\ \mathrm{U}\ q \Rightarrow \mathrm{F}q$
$\models \overset{\infty}{\mathrm{G}}q \Rightarrow \overset{\infty}{\mathrm{F}}q$

The idempotence of F, G, $\overset{\infty}{F}$, and $\overset{\infty}{G}$ are asserted below:

$\models FFp \equiv Fp$
$\models \overset{\infty}{F}\overset{\infty}{F} \equiv \overset{\infty}{F}p$
$\models GGp \equiv Gp$
$\models \overset{\infty}{G}\overset{\infty}{G}p \equiv \overset{\infty}{G}p$

Note: of course, $XXp \equiv Xp$ is not valid. We also have that X commutes with F, G, and U

$\models XFp \equiv FXp$
$\models XGp \equiv GXp$
$\models((Xp)\ U\ (Xq)) \equiv X(p\ U\ q)$

The infinitary modalities $\overset{\infty}{F}$ and $\overset{\infty}{G}$ "gobble up" other unary modalities applied to them:

$\models \overset{\infty}{F}p \equiv X\overset{\infty}{F}p \equiv F\overset{\infty}{F}p \equiv G\overset{\infty}{F}p \equiv \overset{\infty}{F}\overset{\infty}{F}p \equiv \overset{\infty}{G}\overset{\infty}{F}p$
$\models \overset{\infty}{G}p \equiv X\overset{\infty}{G}p \equiv F\overset{\infty}{G}p \equiv G\overset{\infty}{G}p \equiv \overset{\infty}{F}\overset{\infty}{G}p \equiv \overset{\infty}{G}\overset{\infty}{G}p$

(Note: in the above we make use of the abuse of notation that $\models a_1 \equiv ... \equiv a_n$ abbreviates the n-1 valid equivalences $\models a_1 \equiv a_2, ..., \models a_{n-1} \equiv a_n$.) The F, $\overset{\infty}{F}$ operators have an existential nature, the G, $\overset{\infty}{G}$ operators a universal nature, while the U operator is universal in its first argument and existential in its second argument. We thus have the following distributivity relations between these temporal operators and the boolean connectives $\wedge$ and $\vee$:

$\models F(p \vee q) \equiv (Fp \vee Fq)$
$\models \overset{\infty}{F}(p \vee q) \equiv (\overset{\infty}{F}p \vee \overset{\infty}{F}q)$
$\models G(p \wedge q) \equiv (Gp \wedge Gq)$
$\models \overset{\infty}{G}(p \wedge q) \equiv (\overset{\infty}{G}p \wedge \overset{\infty}{G}q)$
$\models((p \wedge q)\ U\ r) \equiv ((p\ U\ r) \wedge (q\ U\ r))$
$\models(p\ U\ (q \vee r)) \equiv ((p\ U\ q) \vee (p\ U\ r))$

Since the X operator refers to a unique next moment, it distributes with all the boolean connectives:

$\models X(p \vee q) \equiv (Xp \vee Xq)$
$\models X(p \wedge q) \equiv (Xp \wedge Xq)$
$\models X(p \Rightarrow q) \equiv (Xp \Rightarrow Xq)$
$\models X(p \equiv q) \equiv (Xp \equiv Xq)$

(Note: $\models X\neg p \equiv \neg Xp$ was given above.)

When we mix operators of universal and existential characters we get the following implications, which again cannot be strengthened to equivalences:

$\models(Gp \vee Gq) \Rightarrow G(p \vee q)$
$\models (\overset{\infty}{G}p \vee \overset{\infty}{G}q) \Rightarrow \overset{\infty}{G}(p \vee q)$
$\models F(p \wedge q) \Rightarrow Fp \wedge Fq$
$\models \overset{\infty}{F}(p \wedge q) \Rightarrow (\overset{\infty}{F}p \wedge \overset{\infty}{F}q)$
$\models ((p\ U\ r) \vee (q\ U\ r)) \Rightarrow ((p \vee q)\ U\ r)$
$\models (p\ U\ (q \wedge r)) \Rightarrow ((p\ U\ q) \wedge (p\ U\ r))$

We next note that the temporal operators below are monotonic in each argument:

$$\models G(p \Rightarrow q) \Rightarrow (Gp \Rightarrow Gq)$$
$$\models G(p \Rightarrow q) \Rightarrow (Fp \Rightarrow Fq)$$
$$\models G(p \Rightarrow q) \Rightarrow (Xp \Rightarrow Xq)$$
$$\models G(p \Rightarrow q) \Rightarrow (\overset{\infty}{F}p \Rightarrow \overset{\infty}{F}q)$$
$$\models G(p \Rightarrow q) \Rightarrow (\overset{\infty}{G}p \Rightarrow \overset{\infty}{G}q)$$
$$\models G(p \Rightarrow q) \Rightarrow ((p\ U\ r) \Rightarrow (q\ U\ r))$$
$$\models G(p \Rightarrow q) \Rightarrow ((r\ U\ p) \Rightarrow (r\ U\ q))$$

Finally, we have following important *fixpoint characterizations* of the temporal operators (cf. Section 8.4):

$$\models Fp \equiv p \lor XFp$$
$$\models Gp \equiv p \land XGp$$
$$\models (p\ U\ q) \equiv q \lor (p \land X(p\ U\ q))$$
$$\models (p\ B\ q) \equiv \neg q \land (p \lor X(p\ B\ q))$$

3.2.5 Minor Variants of PLTL

One minor variation is to change the basic temporal operators. There are a number of variants of the until operator p U q, which is defined as the *strong until*: there does exist a future state where q holds and p holds until then. We could write $p\ U_s\ q$ or $p\ U_\exists\ q$ to emphasize its strong, existential character. The operator *weak until*, written $p\ U_w\ q$ (or $p\ U_\forall q$), is an alternative. It intuitively means that p holds for as long as q does not, even forever if need be. It is also called the *unless* operator. Its technical definition can be formulated as:

$$x \models p\ U_\forall\ q \text{ iff } \forall j\ (\ (\forall k \leq j\ x^k \models \neg q) \text{ implies } x^j \models p\ )$$

exhibiting its "universal" character. Note that, given the boolean connectives, each until operator is expressible in terms of the other:

(a) $p\ U_\exists\ q \equiv p\ U_\forall\ q \land Fq$

(b) $p\ U_\forall\ q \equiv p\ U_\exists\ q \lor Gp \equiv p\ U_\exists\ q \lor G(p \land \neg q)$

We also have variations based on the answer to the question: does the future include the present? The future does include the present in our formulation, and is thus called the *reflexive* future. We might instead formulate versions of the temporal operators referring to the *strict* future, i.e., those times strictly greater than the present. A convenient notation for emphasizing the distinction involves use of $>$ or $\geq$ as a superscript:

$F^{>}p$ — $\exists$ a strict future moment when p holds
$F^{\geq}p$ — $\exists$ a moment, either now or in the future, when p holds
$F^{>}p \equiv XF^{\geq}p$
$F^{\geq}p \equiv p \lor F^{>}p$

Similarly we have the strict always ($G^{>}p$) in addition to our "ordinary" always ($G^{\geq}p$).

The *strict* (strong) until P $U^{>}$ q $\equiv$ X(p U q) is of particular interest. Note that *false* $U^{>}$q $\equiv$ X(false U q) $\equiv$ Xq. The single modality strict, strong until is enough to define all the other linear time operators (as shown by Kamp [Ka68].)

Remark: One other common variation is simply notational. Some authors use $\Box$p for Gp, $\Diamond$p for Fp, and $\circ$p for Xp.

Another minor variation is to change the underlying structure to be any initial segment I of |N, possibly a finite one. This seems sensible because we may want to reason about terminating programs as well as nonterminating ones. We then correspondingly alter the meanings of the basic temporal operators, as indicated (informally) below:

Gp — for all subsequent times in I, p holds.
Fp — for some subsequent times in I, p holds.
p U q — for some subsequent time in I q holds, and p holds at all subsequent times until then.

We also now can distinguish two notions of nexttime:

$X_{\forall}$p—weak nexttime—if there exists a successor moment then p holds there
$X_{\exists}$ p—strong nexttime—there exists a successor moment and p holds there

Note that each nexttime operator is the dual of the other: $X_{\exists}p \equiv (\neg X_{\forall} \neg p$ and $X_{\forall}p \equiv \neg X_{\exists} \neg p)$.

Remark: Without loss of generality, we can restrict our attention to structures where the timeline = |N and still get the effect of finite timelines. This can be done in either of two ways:

(a) Repeat the final state so the finite sequence $s_0\ s_1 \ldots s_k$ of states is represented by the infinite sequence $s_0\ s_1 \ldots s_k\ s_k\ s_k \ldots$. (This is somewhat like adding a self-loop at the end of a finite, directed linear graph.)

(b) Have a proposition P_{GOOD} true for exactly the good (i.e., finite) portion of the timeline.

Adding past tense temporal operators

As used in computer science, all temporal operators are future tense; we might use the following suggestive notation and terminology for emphasis:

F^{+}p — sometime in the future p holds,
G^{+}p — always in the future p holds,
X^{+}p — nexttime p holds (Note: "next" implies implicitly the future)
p U^{+} q — sometime in the future q holds and p holds subsequently until then

However, as originally studied by philosophers there were past tense operators as well; we can use the corresponding notation and terminology:

F^{-}p — sometime in the past p holds
G^{-}p — always in the past p holds
$X_{\exists}^{-}$p — lasttime p holds (Note: "last" implicitly refers to the past)
p U^{-}q — sometime in the past q holds and p holds previously until then

When needed for emphasis we use PLTLF for the logic with just future tense operators, PLTLP for the logic with just past tense operators, and PLTLB for the logic with both.

For temporal logic using the past tense operators, given a linear time structure M = (S,x,L) we interpret formulae over a pair (x,i), where x is the timeline and the natural number index i specifies *where* along the timeline the formula is true. Thus, we write M, (x,i) $\models$ p to mean that "in structure M along timeline x at time i formula p holds true;" when M is understood we write just (x,i) $\models$ p. Intuitively, pair (x,i) corresponds to the suffix x^i, which is the forward interval $x[i{:}\infty)$ starting at time i, used in the definition of the future tense operators. When the past is allowed the pair (x,i) is needed since formulae can reference positions along the entire timeline, both forward and backward of position i. If we restrict our attention to just the future tense as in the definition of PLTL, we can omit the second component of (x,i) – in effect assuming that i=0, and that formulae are interpreted at the beginning of the timeline – and write x $\models$ p for (x,0) $\models$ p.

The technical definitions of the basic past tense operators are as follows:

$(x,i) \models p\ U^{-}q$ iff $\exists j(j \leq i$ and $(x,j) \models q$ and $\forall k\ (j < k \leq i$ implies $(x,j) \models p))$
$(x,i) \models X_{\exists}^{-} p$ iff $i \geq 0$ and $(x,i\text{-}1) \models p$

Note that the lasttime operator is strong, having an existential character, asserting that there is a past moment; thus is false at time 0.

The other past connectives are then introduced as abbreviations as usual: e.g., the weak lasttime $X_{\forall}^{-} p$ for $\neg X_{\exists}^{-}\ \neg p$, $F^{-}p$ for (*true* $U^{-}p$), and $G^{-}p$ for $\neg F^{-}\ \neg p$.

For comparison we also present the definitions of some of the basic future tense operators using the pair (x,i) notation:

$(x,i) \models (p\ U\ q)$ iff $\exists j\ (j \geq i$ and $(x,j) \models q$ and $\forall k(i \leq k < j$ implies $(x,k) \models p))$
$(x,i) \models Xp$ iff $(x,i+1) \models p$
$(x,i) \models Gq$ iff $\forall j(j \geq i$ implies $(x,j) \models q)$
$(x,i) \models Fq$ iff $\exists j(j \geq i$ and $(x,j) \models q)$

Remark: Philosophers used a somewhat different notation. $F^{-}p$ was usually written as Pp, $G^{-}p$ as Hp, and p U q as p S q meaning "p since q." We prefer the present notation due to its more uniform character.

The decision whether to allow i to float or to anchor it at 0 yields different notions of equivalence, satisfiability, and validity. We say that a formula p is *initially satisfiable* provided there exists a linear time structure M = (S,x,L) such that M,(x,0) $\models$ p. We say that a formula p is *initially valid* provided for all timeline structures M = (S,x,L) we have M,(x,0) $\models$ p. We say that a formula p is *globally satisfiable* provided that there exists a linear time structure M = (S,x,L) and time i such that M,(x,i) $\models$ p. We say that a formula p is *globally valid* provided that for all linear time structures M = (S,x,L) and times i we have M,(x,i) $\models$ p.

In an almost trivial sense inclusion of the past tense operators increases the expressive power of our logic:

We say that formula p is *globally equivalent* to formula q, and write $p \equiv_g q$, provided that $\forall$ linear structure x $\forall$ time $i \in |N$ $[(x,i) \models p$ iff $(x,i) \models q]$.

Theorem 3.1. As measured with respect to global equivalence, PLTLB is strictly more expressive than PLTLF.

Proof. The formula $F^{-}Q$ is not expressible in PLTLF, as can be seen by considering two structures x, x′ as depicted below.

$$\begin{tikzcd}
\overset{Q}{\bullet} \arrow[r] & \bullet \arrow[r, squiggly] & {} \\
\overset{\neg Q}{\bullet} \arrow[r] & \bullet \arrow[r, squiggly] & {}
\end{tikzcd}$$

The structures are identical except for their respective state at time 0. At time 1 $F^{-}Q$ distinguishes the two structures (i.e. $(x,1) \models F^{-}Q$ and $(x',1) \not\models F^{-}Q$) yet future tense PLTLF cannot distinguish (x,1) from (x′,1), since the infinite suffixes beginning at time 1 are identical. $\square$

Yet in the sense that programs begin execution in an initial state, inclusion of the past tense operators adds no expressive power.

We say that formula p is *initially equivalent* to formula q, and write $p \equiv_i q$, provided that $\forall$ linear structure x $[(x,0) \models p$ iff $(x,0) \models q]$.

Theorem 3.2. As measured with respect to initial equivalence, PLTLB is equivalent in expressive power to PLTLF.

This can be proved using results regarding the theory of linear orderings (cf. [GPSS80]):

We also note the following relationship between $\equiv_i$ and $\equiv_g$:

Proposition 3.3. $p \equiv_g q$ iff $Gp \equiv_i Gq$.

By convention we shall take satisfiable to mean initially satisfiable and valid to mean initially valid, unless otherwise stated. Intuitively, this makes sense since programs start execution in an initial state. Moreover, whenever we refer to expressive power we are measuring it with respect to initial equivalence, unless otherwise stated. One benefit of comparing expressive power on the basis of initial equivalence, is that it suggests we view formulae of PLTL and its variants as defining sets of sequences, i.e. formal languages. (See section 6.)

3.3 First-Order Linear Temporal Logic (FOLTL)

First-order linear temporal logic (FOLTL) is obtained by taking propositional linear temporal logic (PLTL) and adding to it a First order language **L**. That is, in addition to atomic propositions, truth-functional connectives, and temporal operators we now also have predicates, functions, individual constants, and individual variables, each interpreted over an appropriate domain with the standard Tarskian definition of truth.

Symbols of L

We have a first order language **L** over a set of *function* symbols and a set of *predicate* symbols. The zero-ary function symbols comprise the subset of *constant* symbols. Similarly, the zero-ary predicate symbols are known as the *proposition* symbols. Finally, we have a set of individual *variable* symbols.

We use the following notations:

ϕ, ψ, ..., etc. for n-ary, $n \geq 1$, predicate symbols,
P, Q, ..., etc. for proposition symbols,
f, g, ..., etc. for n-ary, $n \geq 1$, function symbols,
c, d, ..., etc. for constant symbols, and
y, z, ..., etc. for variable symbols.

We also have the distinguished binary predicate symbol $\approx$, known as the *equality symbol*, which we use in the standard infix fashion. Finally, we have the usual quantifier symbols $\forall$ and $\exists$, denoting universal and existential quantification, respectively, which are applied to individual variable symbols, using the usual rules regarding scope of quantifiers, and free and bound variables.

Syntax of L

The *terms* of **L** are defined inductively by the following rules:

T1 Each constant c is a term.

T2 Each variable y is a term.

T3 If $t_1, \ldots, t_n$ are terms and f is an n-ary function symbol then $f(t_1, \ldots, t_n)$ is a term.

The *atomic formulae* of **L** are defined by the following rules:

AF1 Each 0-ary predicate symbol (i.e. atomic proposition) is an atomic formula.

AF2 If $t_1, \ldots, t_n$ are terms and ψ is an n-ary predicate then $\psi(t_1, \ldots, t_n)$ is an atomic formula.

AF3 If t_1, t_2 are terms then $t_1 \approx t_2$ is also an atomic formula.

Finally, the (compound) formulae of **L** are defined inductively as follows:

F1 Each atomic formula is a formula.

F2 If p,q are formulae then $(p \wedge q)$, $\neg p$ are formulae.

F3 If p is a formula and y is a free variable in p then $\exists y p$ is a formula.

Semantics of L

The semantics of **L** is provided by an interpretation I over some domain D. The interpretation I assigns an appropriate meaning over D to the (non-logical) symbols of **L**: Essentially, the n-ary predicate symbols are interpreted as concrete, n-ary relations over D, while the n-ary function symbols are interpreted as concrete, n-ary functions on D. (Note: an n-ary relation over D may be viewed as an n-ary function $D^n \rightarrow |B$, where $|B = \{$*true*, *false*$\}$ is the distinguished Boolean domain.) More precisely I assigns a meaning to the symbols of **L** as follows:

- for an n-ary predicate symbol ψ, $n \geq 1$, the meaning $I(\psi)$ is a function $D^n \rightarrow |B$
- for a proposition symbol P, the meaning I(P) is an element of |B
- for an n-ary function symbol f, $n \geq 1$, the meaning I(f) is a function $D^n \rightarrow D$
- for an individual constant symbol c, the meaning I(c) is an element of D
- for an individual variable symbol y, the meaning I(y) is an element of D

The interpretation I is extended to arbitrary terms, inductively:

$I(f(t_1, \ldots, t_n)) = I(f)\ (I(t_1), \ldots, I(t_n))$

We now define the meaning of truth under interpretation I of formula p, written $I \models p$. First, for atomic formulae we have:

$I \models P$, where P is an atomic proposition, iff $I(P) =$ *true*.
$I \models \psi(t_1, \ldots, t_n)$, where ψ is an n-ary predicate and $t_1, \ldots, t_n$ are terms,
iff $I(\psi)\ (I(t_1), \ldots, I(t_n)) =$ *true*.
$I \models t_1 \approx t_2$ iff $I(t_1) = I(t_2)$.

Next, for compound formulae we have:

$I \models p \wedge q$ iff $I \models p$ and $I \models q$.
$I \models \neg p$ iff it is not the case that $I \models p$.
$I \models \exists y\ p$, where y is a free variable in p, iff there exists some $d \in D$ such that $I[y \leftarrow d] \models p$ where $I[y \leftarrow d]$ is the interpretation identical to I except that y is assigned value d.

Global versus Logical Symbols

For defining First Order Linear Temporal Logic (FOLTL), we assume that the set of symbols is divided into two classes, the class of *global* symbols and the class of *local* symbols. Intuitively, each global symbol has the same interpretation over all states; the interpretation of a local symbol may vary, depending on the state at which it is evaluated. We will subsequently assume that all function symbols (and thus all constant symbols) are global, and that all n-ary predicate symbols, for $n \geq 1$, are also global. Proposition symbols (i.e. 0-ary predicate symbols) and variable symbols may be local or may be global.

A *(first order) linear time structure* M = (S,x,L) is defined just as in the propositional case, except that L now associates with each state s an interpretation L(s) of all symbols at s, such that for each global symbol w, $L(s)(w) = L(s')(w)$, for all $s,s' \in S$. Note that the structure M has an underlying domain D, as for **L**. Also, it is sometimes convenient to refer to the global interpretation I associated with M by I(w) = L(s)(w), where w is any global symbol and s is any state of M. (Note: Implicitly given with a structure is its *signature* or *similarity type* consisting of the alphabets of all the different kinds of symbols. The signature of the structure is assumed to match that of the language (FOLTL).)

Description of FOLTL

We are now ready to define the language of First-Order Linear Temporal Logic (FOLTL) obtained by adding **L** to PLTL. First, the terms of FOLTL are those generated by rules T1-3 for **L** plus the rule:

T4 If t is a term then Xt is a term (intuitively, denoting the immediate future value of term t).

The atomic formulae of FOLTL are generated by the same rules as for **L**, but now are used in conjunction with the expanded set of rules T1-4 for terms.

Finally, the (compound) formulae of FOLTL are defined inductively using the following rules:

FOLTL1 Each atomic formula is a formula.

FOLTL2 If p,q are formulae, then so are $p \wedge q$, $\neg p$.

FOLTL3 If p,q are formulae, then so are p U q, Xp.

FOLTL4 If p is a formula and y is a free variable in p, then $\exists y$ p is a formula.

The semantics of FOLTL is provided by a first order linear time structure M over a domain D as above. Global interpretation I of M assigns a meaning to each global symbol, while the local interpretations L(–) associated with M assign a meaning to each local symbol.

Since the terms of FOLTL are generated by rules T1-3 for **L** plus the rule T4 above, we extend the meaning function—now denoted by a pair (M,x)—for terms:

(M,x) (c) = I(c), since all constants are global.
(M,x) (y) = I(y), where y is a global variable.
(M,x) (y) = $L(s_0)$ (y), where y is a local variable and x = $(s_0,s_1,s_2,\ldots)$.
(M,x) $(f(t_1, \ldots, t_n))$ = (M,x) (f) ((M,x) (t_1), ..., (M,x) (t_n)).
(M,x) (Xt) = (M,x^1) (t).

Now the extension of $\models$ is routine. For atomic formulae we have:

M,x $\models$ P iff I $\models$ P where P is a global proposition.
M,x $\models$ P iff $L(s_0)$ (P) = *true* where P is a local proposition and x = $(s_0,s_1,s_2,\ldots)$.
M,x $\models \psi(t_1,\ldots,t_n)$ iff (M,x) (ψ) ((M,x) (t_1),..., (M,x) (t_n)) = *true*.
M,x $\models t_1 \approx t_2$ iff (M,x) (t_1) = (M,x) (t_2).

We finish off the semantics of FOLTL with the inductive definition of $\models$ for compound formulae:

M,x $\models p \wedge q$ iff M,x $\models$ p and M,x $\models$ q.

M,x $\models \neg p$ iff it is not the case that M,x $\models$ p.

M,x $\models$ (p U q) iff $\exists$j (M,x^j $\models$ q and $\forall$k<j (M,x^k $\models$ p))

M,x $\models$ Xp iff M,x^1 $\models$ p

M,x $\models$ $\exists$y p, where y is a global variable free in p, iff there exists some d $\in$ D for which M[y $\leftarrow$ d],x $\models$ p, where M[y $\leftarrow$ d] is the structure having global interpretation I[y $\leftarrow$ d] identical to I except y is assigned the value d.

A formula p of FOLTL is *valid* iff for every for every first order linear time structure M = (S,x,L) we have M,x $\models$ p. The formula p is *satisfiable* iff there exists M = (S,x,L) such that M,x $\models$ p.

Remark. For notational simplicity we have assumed the **L** is a one-sorted first order language. Thus each symbol (function symbol, predicate symbol, etc.) is of the same sort and is interpreted over the single domain D. For certain applications, it is more convenient to assume that **L** is a multi-sorted language, where the symbols of **L** are partitioned into different sets, each of which corresponds to a different domain with different argument positions. The extension to multi-sorted languages is routine, although a bit cumbersome notationally.

4 The Technical Framework of Branching Temporal Logic

4.1 Tree-like Structures

In branching time temporal logics, the underlying structure of time is assumed to have a branching tree-like nature where each moment may have many successor moments. The structure of time thus corresponds to an infinite tree. In the sequel, we will further assume that along each path in the tree, the corresponding time line is isomorphic to |N. We do allow a node in the tree to have infinitely many (even uncountably many) successors, while we require each node to have at least one successor. It will turn out that, as far as our branching temporal logics are concerned, such trees are indistinguishable from trees with finite, even bounded, branching. Trees of the latter type have a natural correspondence with the computations of concurrent or nondeterministic programs, as discussed in the next section.

We say that a *temporal structure* M = (S,R,L) where

S is the set of *states*,

R is a total *binary relation* $\subseteq$ S $\times$ S (i.e., one where $\forall$s $\in$ S $\exists$t $\in$ S (s,t) $\in$ R), and

L:S $\rightarrow$ PowerSet(AP) is a labelling which associate with each state s an interpretation L(s) of all atomic proposition symbols at state s.

We may view M as a labelled, directed graph with node set S, arc set R, and node labels given by L. We say (the graph of) M

(a) is *acyclic* provided it contains no directed cycles;

(b) is *tree-like* provided that it is acyclic and each node has at most 1 R-predecessor (i.e., there is no "merging" of paths); and

(c) is a *tree* provided that it is tree-like and there exists a unique node—called the *root*—from which all other nodes of M are reachable and that has no R-predecessors.

We have not required that (the graph of) M be a tree. However, we may assume, without loss of generality, that it is. We define the Structure $\hat{M} = (\hat{S},\hat{R},\hat{L})$, called the structure obtained by unwinding M starting at state $s_0 \in S$, where $\hat{S}$, $\hat{R}$ are, respectively, the least subsets of $S \times |N$, $\hat{S} \times \hat{S}$ such that:

- $(s_0,0) \in \hat{S}$
- if $(s,n) \in \hat{S}$ then
  $\{(t,n+1) : t$ is an R-successor of s in M$\} \subseteq \hat{S}$, and
  $\{((s,n), (t,n+1)) : t$ is an R-successor of s in M$\} \subseteq \hat{R}$;

and $\hat{L}((s,n)) = L(s)$. Then (the graph of) $\hat{M}$ is a tree with root $(s_0,0)$, and it is easily checked that, for all the branching time logics we will consider, a formula p holds at s_0 in M iff p holds at $(s_0,0)$ in $\hat{M}$. See Figure 2.

4.2 Propositional Branching Temporal Logics

In this section we provide the formal syntax and semantics for two representative systems of propositional branching time temporal logics The simpler logic, CTL (Computational Tree Logic) allows basic temporal operators of the form: a path quantifier—either A ("for all futures") or E ("for some future"—followed by a single one of the usual linear temporal operators G ("always"), F ("sometime"), X ("nexttime"), or U ("until"). It corresponds to what one might naturally first think of as a branching time logic. CTL is closely related to branching time logics proposed in [La80], [EC80], [QS81], [BPM81], and was itself proposed in [CE81]. However, as we shall see, its syntactic restrictions significantly limit its expressive power. We therefore also consider the much richer language CTL*, which is sometimes referred to informally as full branching time logic. The logic CTL* extends CTL by allowing basic temporal operators where the path quantifier (A or E) is followed by an arbitrary linear time formula, allowing boolean combinations and nestings, over F, G, X, and U. It was proposed as a unifying framework in [EH86], subsuming both CTL and PLTL, as well as a number of other systems. Related systems of high expressiveness are considered in [Pa79], [Ab80], [ST81], and [VW83].

Syntax

We now give a formal definition of the syntax of CTL*. We inductively define a class of state formulae (true or false of states) using rules S1-3 below and a class of path formulae (true or false of paths) using rules P1-3 below:

S1 Each atomic proposition P is a state formula

S2 If p,q are state formulae then so are $p \wedge q$, $\neg p$

S3 If p is a path formula then Ep, Ap are state formulae

P1 Each state formula is also a path formula

P2 If p,q are path formulae then so are $p \wedge q$, $\neg p$

P3 If p,q are path formulae then so are Xp, p U q

The set of state formulae generated by the above rules forms the language CTL*. The other connectives can then be introduced as abbreviations in the usual way.

Remark: We could take the view that Ap abbreviates $\neg E \neg p$, and give a more terse syntax in terms of just the primitive operators E, $\wedge$, $\neg$, X, and U. However, the present approach makes it easier to give the syntax of the sublanguage CTL below.

The restricted logic CTL is obtained by restricting the syntax to disallow boolean combinations and nestings of linear time operators. Formally, we replace rules P1-3 by

P0 if p,q are state formulae then Xp, p U q are path formulae.

The set of state formulae generated by rules S1-3 and P0 forms the language CTL. The other boolean connectives are introduced as above while the other temporal operators are defined as abbreviations as follows: EFp abbreviates E(*true* U p), AGp abbreviates $\neg EF \neg p$, AFp abbreviates A(*true* U p), and EGp abbreviates $\neg AF \neg p$. (Note: this definition can be seen to be consistent with that of CTL*.)

Also note that the set of path formulae generated by rules by P1-P3 yield the linear time PLTL.

Semantics

A formula of CTL* is interpreted with respect to a structure M = (S,R,L) as defined above. A *fullpath* of is an infinite sequence $s_0, s_1, s_2, \ldots$ of states such that $\forall i \ (s_i, s_{i+1}) \in R$. We use the convention that $x = (s_0, s_1, s_2, \ldots)$ denotes a fullpath, and that x^i denotes the suffix path $(s_i, s_{i+1}, s_{i+2}, \ldots)$. We write $M, s_0 \models p$ (respectively, $M, x \models p$) to mean that state formula p (respectively, path formula p) is true in structure M at state s_0 (respectively, of fullpath x). We define $\models$ inductively as follows:

S1 $M, s_0 \models p$ iff $P \in L(s_0)$

S2 $M, s_0 \models p \wedge q$ iff $M, s_0 \models p$ and $M, s_0 \models q$ $M, s_0 \models \neg p$ iff not$(M, s_0 \models p)$

S3 $M, s_0 \models Ep$ iff $\exists$ fullpath $x = (s_0, s_1, s_2, \ldots)$ in M, $M, x \models p$
$M, s_0 \models Ap$ iff $\forall$ fullpath $x = (s_0, s_1, s_2, \ldots)$ in M, $M, x \models p$

P1 $M, x \models p$ iff $M, s_0 \models p$

P2 $M, x \models p \wedge q$ iff $M, x \models p$ and $M, x \models q$
$M, x \models \neg p$ iff not$(M, x \models \neg p)$

P3 $M, x \models p \ U \ q$ iff $\exists i \ [M, x^i \models q$ and $\forall j \ (j < i$ implies $M, x^j \models p)]$
$M, x \models Xp$ iff $M, x^1 \models p$

A formula of CTL is also interpreted using the CTL* semantics, using rule P3 for path formulae generated by rule P0.

We say that a state formula p (resp., path formula p) is *valid* provided that for every structure M and every state s (resp., fullpath x) in M we have M,s $\models$ p (resp., M,x $\models$ p). A state formula p (resp., path formula p) is *satisfiable* provided that for some structure M and some state s (resp., fullpath x) in M we have M,s $\models$ p (resp., M,x $\models$ p).

Generalized Semantics

We can define CTL* and other logics over various generalized notions of structure. For example, we could consider more general structures M = (S,X,L) where S is a set of states and L a labelling of states as usual, while $X \subseteq S^\omega$ is a family of infinite computation sequences (fullpaths) over S. The definition of CTL* semantics carries over directly, with path quantification restricted to paths in X, provided that "a fullpath x in M" is understood to refer to a fullpath x in X.

In the most general case X can be completely arbitrary. However, it is often helpful to impose certain requirements on X (cf. [La80], [Pr79], [Ab80], [Em83]). We say that X is *suffix closed* provided that if computation $s_0s_1s_2... \in X$, then the suffix $s_1s_2... \in X$. Similarly, X is *fusion closed* provided that whenever x_1sy_1, $x_2sy_2 \in X$ then $x_1sy_2 \in X$. The idea is that the system should always be able to follow the prefix of one computation and then continue along the suffix sy_2 of another computation; thus the computation actually followed is the "fusion" of two others. Both suffix and fusion closure are needed to ensure that the future behavior of a program depends only on the current state and not how the state is reached.

We may also wish to require that X be *limit closed* meaning that whenever x_1y_1, $x_1x_2y_2$, $x_1x_2x_3y_3$,... are all elements of X, then the infinite path $x_1x_2x_3$..., which is the limit of the prefixes x_1,x_1,x_2,$x_1x_2x_3$..., is also in X. In short, if it possible follow a path arbitrarily long, then it can be followed forever. Finally, a set of paths is *R-generable* if there exists a total binary relation R on S such that a sequence $x = s_0s_2s_2... \in X$ iff $\forall i\ (s_i,s_{i+1}) \in R$. It can be shown that X is R-generable iff it is limit closed, fusion closed and suffix closed. Of course, the basic type of structures we ordinarily consider are R-generable, which correspond to the execution of a program under pure nondeterministic scheduling.

Some such restrictions on the set of paths X are usually needed in order to have the abstract, computation path semantics reflect the behavior of actual concurrent programs. An additional advantage of these restrictions is that they ensure the validity of many commonly accepted principles of temporal reasoning. For example, fusion closure is needed to ensure that EFEFp $\equiv$ EFp. Suffix closure is needed for EFp $\wedge \neg$p $\Rightarrow$ EXEFp, and limit closure for p $\wedge$ AGEXp $\Rightarrow$ EGp. An R-generable structure satisfies all these natural properties.

Another generalization is to define a *multiprocess temporal structure*, which is a refinement of the notion of a branching temporal structure that distinguishes between different processes. Formally, a multiprocess temporal structure M = (S,**R**,L) where

S is a set of states,

R is a finite family $\{R_1,\ldots,R_k\}$ of binary relations R_i on S (intuitively, R_i represents the transitions of process i) such that $R = \cup\, \mathbf{R}$ is total (i.e. $\forall s \in S\ \exists t \in S\ (s,t) \in R$), L associates with each state an interpretation of symbols at the state.

Just as for a (uniprocess) temporal structure, a multiprocess temporal structure may be viewed as a directed graph with labelled nodes and arcs. Each state is represented by a node that is labelled by the atomic propositions true there, and each transition relation R_i is represented by a set of arcs that are labelled with index i. Since there may be multiple arcs labelled with distinct indices between the same pair of nodes, technically the graph-theoretic representation is a directed multigraph.

The previous formulation of CTL* over uniprocess structures refers only to the atomic formulae labelling the nodes. However, it is straightforward to extend it to include, in effect, arc assertions indicating which process performed the transition corresponding to an arc. This extension is needed to formulate the technical definitions of fairness in the next section, so we briefly describe it.

Now, a fullpath $x = (s_0,d_1,s_1,d_2,s_2,\ldots)$, depicted below

$$\begin{tikzcd} \underset{s_0}{\bullet} \arrow[r, "d_1"] & \underset{s_1}{\bullet} \arrow[r, "d_2"] & \underset{s_2}{\bullet} \arrow[r, "d_3"] & \underset{s_3}{\bullet} \arrow[r, "d_4"] & \ldots \end{tikzcd}$$

is an infinite sequence of states s_i alternating with relation indices d_{i+1} such that $(s_i,s_{i+1}) \in R_{d_{i+1}}$, indicating that process d_{i+1} caused the transition from s_i to s_{i+1}. We also assume that there are distinguished propositions $\textit{enabled}_1, \ldots, \textit{enabled}_k$, $\textit{executed}_1, \ldots \textit{executed}_k$, where intuitively $\textit{enabled}_j$ is true of a state exactly when process j is enabled, i.e., when a transition by process j is possible, and $\textit{executed}_j$ is true of a transition when it is performed by process j. Technically, each $\textit{enabled}_j$ is an atomic proposition—and hence a state formula—true of exactly those states in domain R_j:

$$M,s_0 \models \textit{enabled}_j \text{ iff } s_0 \in \text{domain } R_j = \{ s \in S : \exists t \in S\ (s,t) \in R \}$$

while each $\textit{executed}_j$ is an atomic arc assertion—and a path formula such that

$$M,x \models \textit{executed}_j \text{ iff } d_1 = j.$$

It is worth pointing out that there are alternative formalisms that are essentially equivalent to this notion of a (multiprocess) structure. A *transition system* M is a formalism equivalent to a multi-process temporal structure consisting of a triple $M = (S,\mathbf{R},L)$ where $\mathbf{R}$ is a finite family of transitions $\tau_i : S \rightarrow \text{PowerSet}(S)$. To each transition τ_i there is a corresponding relation $R_i = \{(s,t) \in S \times S\text{: } t \in \sigma_i\}$ and conversely. Similarly, there is a correspondence between multiprocess temporal logic structures and **do-od** programs (cf. [Di76]). Assume we are given a **do-od** program $\rho =$ **do** $B_1 \rightarrow A_1$ [] ... [] $B_k \rightarrow A_k$ **od**, where each B_i may be viewed as subset of the state space S and each A_i as a function $S \rightarrow S$. Then we may define an equivalent structure $M=(S,\mathbf{R},L)$, where each $R_i = \{ (s,t) \in S\text{: } s \in B_i \text{ and } t = A_i(s)\}$, and L gives appropriate meanings to the symbols in the program. Conversely, given a structure M, there is a corresponding generalized **do-od** program ρ, where by generalized we mean that each action A_i is allowed to be a relation; viz., it is **do** $B_1 \rightarrow A_1$ []...[] $B_k \rightarrow A_k$ **od**, where each $B_i = \text{domain } R_i = \{ s \in S\text{: } \exists\ t \in S\ (s,t) \in R_i\}$ and $A_i = R_i$.

We can define a single type of general structure which subsumes all of those above. We assume an underlying set of symbols, divided into global and local subsets as before and called *state symbols* to emphasize that they are interpreted over states, as well as an additional set of *arc assertion* *symbols* that are interpreted over transitions (s,t) $\in$ R. Typically we think of L((s,t)) as the set of indices (or names) of processes which could have performed the transition (s,t). A (*generalized*) *fullpath* is now a sequence of states s_i alternating with arc assertions d_i as depicted above.

Now we say that a *general structure* M = (S,R,X,L) where

S is a set of *states*,

R is a total *binary relation* $\subseteq$ S $\times$ S,

X is a set of *fullpaths* over R, and

L is a mapping associating with each state s an interpretation L(s) of all state symbols at s, and with each transition (s,t) $\in$ R an interpretation of each arc assertion at (s,t)

There is no loss of generality due to including R in the definition: for any set of fullpaths X, let R = {(s,t) $\in$ S $\times$ S: there is a fullpath of the form ystz in X, where y is a finite sequence of states and z an infinite sequence of states in S}; then all consecutive pairs of states along paths in X are related by R.

The extensions needed to define CTL* over such a general structure M are straightforward. The semantics of path quantification as specified in rule S3 carries over directly to the general M, provided that a “full path in M” refers to one in X. If d is an arc assertion we have that:

M,x $\models$ d iff d $\in$ L((s_0,s_1))

4.3 First-Order Branching Temporal Logic

We can define systems of First-order Branching Temporal Logic. The syntax is obtained by combining the rules for generating a system of propositional Branching Temporal Logic plus a (multi-sorted) first-order language. The underlying structure M = (S,R,L) is extended so that it associates with each state s an interpretation L(s) of local and global symbols at state s, including in particular local variables as well as local atomic propositions. The semantics is given by the usual Tarskian definition of truth. Validity and satisfiability are defined in the usual way. The details of the technical formulation are closely analogous to those for first-order linear temporal logic and are omitted here.

5 Concurrent Computation: A Framework

5.1 Modelling Concurrency by Nondeterminism and Fairness

Our treatment of concurrency is the usual one where concurrent execution of a system of processes is modelled by the nondeterministic interleaving of atomic actions of the individual processes. The semantics of a concurrent program is thus given by a computation tree: a concurrent program starting in a given state may follow any one of a (possibly infinite) number of different computation paths in the tree (i.e., sequences of execution states) corresponding to the different sequences of nondeterministic choices the program might make. Alternatively, the semantics can be given simply by the set of all possible execution sequences, ignoring that they can be organized into a tree, for each possible starting state.

We remark that it is always possible to model concurrency by nondeterminism, since by picking a sufficiently fine level of granularity for the atomic actions to be interleaved, any behavior that could be produced by true concurrency (i.e., true simultaneity of action) can be simulated by interleaving. In practice, it is helpful to use as coarse of granularity as possible, as it reduces the number of interleavings that must be considered.

There is one additional consideration in the modelling of concurrency by nondeterminism. This is the fundamental notion of *fair scheduling assumptions*, commonly called *fairness*, for short.

In a truly concurrent system, implemented physically, it would be reasonable to assume that each sequential process P_i of a concurrent program $P_1 \| \ldots \| P_n$ is assigned to its own physical processor. Depending on the relative rates of speed at which the physical processors ran, we would expect that the corresponding nondeterministic choices modeling this concurrent system, would favor, more often the faster processes. For a very simple example, consider a system $P_1 \| P_2$ with just two processes. If each process ran on its own physical processor, and the processors ran at approximately equal speeds, we would expect the corresponding sequence of interleavings of steps of the individual processes to be of the form:

$P_1P_2P_1P_2P_1P_2\ldots$

or

$P_2P_1P_2P_1P_2P_1\ldots$

or, perhaps

$P_1P_1P_2P_1P_2P_2P_1P_1P_2\ldots$

where, for each i, after i steps in all have been executed, roughly i/2 steps of each individual process has been executed. If processor 1 ran, say, three times faster than processor 2 we would expect corresponding interleavings such as

$P_1P_1P_1P_2P_1P_1P_1P_2P_1P_1P_1P_2\ldots$

where steps of process P_1 occur about 3 times more often than steps of process P_2.

Now, on the other hand, we would not expect to see a sequence of actions such as $P_1P_1P_1P_1$ ...where process P_1 is always chosen while process P_2 is never chosen. This would be *unfair* to process P_2. Under the assumption that each processor is always running at some positive, finite speed, regardless of how the relative ratios of the processor's speed might vary, we would thus expect to see *fair* sequences of interleavings where each process is executed infinitely often. This notion of fair scheduling thereby corresponds to the reasonable and very weak assumption that each process makes some progress. In the sequel, we shall assume that the nondeterministic choices of which process is to next execute a step are such that resulting infinite sequence is fair.

For the present we let the above notion of fairness—that each process be executed infinitely often—suffice; actually, however, there are a number of technically distinct refinements of this notion. (See, for example, the book by Francez [Fr86] as well as [Ab80], [FK84], [GPSS80], [La80], [LPS81], [Pn83], [QS83], [LPZ85] and [EL85].) Some of these will be described subsequently.

Thus to model the semantics of concurrency accurately we need fairness assumptions in addition to the computation sequences generated by nondeterministic interleaving of the execution of individual processes.

We remark on an advantage afforded by fairness assumptions. By the principal of separation of concerns, we should distinguish the issue of correctness of a program, from concerns with its efficiency or performance. Correctness is a qualitative sort of property. To say that we are concerned that a program be totally correct means we wish to establish that it does eventually terminate meeting a certain post condition. Establishing just when it terminates is a quantitative sort of property that is distinct from the qualitative notion of eventually terminating. Temporal logic is especially appropriate for such qualitative reasoning. Moreover, fairness assumptions facilitate such qualitative reasoning. Since fairness corresponds to the very weak qualitative notion that each process is running at some finite positive speed, programs proved correct under a fair scheduling assumption will be correct no matter what the rates are at which the processors actually run.

We very briefly summarize the preceding discussion by saying that, for our purposes, concurrency = nondeterminism + fairness. Somewhat less pithily but more precisely and completely, we can say that a concurrent program amounts to a global state transition system, with global state space essentially the cartesian product of the state spaces of the individual sequential processes and transitions corresponding to the atomic actions of the individual sequential processes, plus a fairness constraint and a starting condition. The behavior of a concurrent program is then described in terms of the trees (or simply sets) containing all the computation sequences of the global state transition system which meet the fair scheduling constraint and starting condition.

5.2 Abstract Model of Concurrent Computation

With the preceding motivation, we are now ready to describe our abstract model of concurrent computation.

An *abstract concurrent program* is a triple (M, ϕ_{START}, Φ) where M is a (multiprocess) temporal structure, ϕ_{START} is an atomic proposition corresponding to a distinguished set of starting states in M, Φ is a fair scheduling constraint which we, for convenience, take to be specified in linear temporal logic.

Among possible fairness constraints, are the following very common ones:

(1) *Impartiality*: An infinite sequence is impartial iff every process is executed infinitely often during the computation, which is expressed by $\Phi = \wedge_{i=1}^{k} \overset{\infty}{\mathrm{F}}$ *executed*$_i$

(2) *Weak fairness* (also known as *justice*): An infinite computation sequence is *weakly fair* iff every process enabled almost everywhere is executed infinitely often, which is expressed by $\Phi = \wedge_{i=1}^{k} (\overset{\infty}{\mathrm{G}}\ enabled_i \Rightarrow \overset{\infty}{\mathrm{F}}\ executed_i)$

(3) *Strong fairness* (also known simply as *fairness*): An infinite computation sequence is strongly fair iff every process enabled infinitely often is executed infinitely often, which is expressed by $\Phi = \wedge_{i=1}^{k} (\overset{\infty}{\mathrm{F}}\ enabled_i \Rightarrow \overset{\infty}{\mathrm{F}}\ executed_i)$

5.3 Concrete Models of Concurrent Computation

Different concrete models of concurrent computation can be obtained from our abstract model by refining it in various ways. These include:

(i) providing structure for the global state space,

(ii) defining (classes of) instructions which each process can execute to manipulate the state space, and

(iii) providing concrete domains for the global state space.

We now describe some concrete models of concurrent computation.

Concrete Models of Parallel Computation Based on Shared Variables

Here, we consider parallel programs of the form $P_1 \| P_2 \| \ldots \| P_k$ consisting of a finite, fixed set of sequential processes $P_1,\ldots,P_k$ running together in parallel. There is also an underlying set of *variables* $v_1,\ldots,v_m$ assuming values in a domain D, that are *shared* among the processes in order to provide for inter-process communication and coordination. Thus, the global state set S consists of tuples of the form $(l_1,\ldots,l_k,v_1,\ldots,v_m) \in \times_{h=1}^{k} LOC(P_h) \times \times_{h'=1}^{m} D_{h'}$, where each process P_i has an associated set $LOC(P_i) = \{l_i^1,\ldots, l_i^{n_i}\}$ of locations. Each process P_i is described by a transition diagram with nodes labelled by locations. Alternatively, a process can be described by an equivalent text. Associated with each arc (l,l′) there is an *instruction* I which may be executed by process P_i whenever process P_i is selected for execution and the current global state has the location of P_i at l. The instruction I is presented as a guarded command $B \rightarrow A$, where guard B is a predicate over the variables $\bar{v}$ and action A is an assignment $\bar{u} := \bar{e}$ of a tuple of expressions to the corresponding tuple of variables.

It is possible to make further refinements of the model. By imposing appropriate restrictions on the way instructions can access (i.e., read) and manipulate (i.e., write) the data we can get models ranging from those that can perform "test-and-set" instructions which permit a read followed by a write in a single atomic operation on a variable to those that only permit an atomic read or an atomic write of a variable.

We might also wish to impose restrictions on which processes are allowed which kind of access to which variables. One such rule is that each variable v is "owned" by some one unique process P, (think of v as being in the "local" memory of process p); then, each process can read any variable in the system, while only the process which owns a variable can write into it. This specialization is referred to as the *distributed shared-variables* model.

Still, another refinement is to specify a specific domain for the variables, say |N = the natural numbers. Yet another is to specify the type of instructions (e.g. "copy the value of variable y into variable z"). They can be combined to get a completely concrete program with instructions such as "load the value of variable z into variable y and decrement by the natural number 1."

Concrete Models of Parallel Computation based on Message Passing

This model is similar to the previous one. However, each process has its own set of local variables $y_1,\ldots,y_n$ that cannot be accessed by other processes. All interprocess communication is effected by message passing primitives similar to those of CSP [Ho78]; processes communicate via *channels*, which are essentially message buffers of length 0. The communication primitives are

- B;e!α—send the value of expression e along channel α, provided that guard predicate B is enabled and there is a corresponding receive command ready.
- B;v?α—receive a value along channel α and store it in variable v, provided that the guard predicate B is enabled and there is a corresponding send command ready.

As in CSP, we assume that message transmission occurs as a single, synchronous event, with sender and receiver simultaneously executing the send, resp. receive primitive.

Remark. For programs in one of the above concrete frameworks, we use atomic propositions such as atl_i^j to indicate that, in the present state, process i is at location l_j.

5.4 Connecting the Concurrent Computation Framework with Temporal Logic

For an abstract concurrent program (M,ϕ_{START},Φ) and Temporal Logic formula p we write (M,ϕ_{START},Φ) $\models$ p and read it precisely (and a bit long-windedly) as "for program text M with starting condition ϕ_{START} and fair scheduling constraint Φ, formula p holds true;" the technical definition is as follows.

(i) in the linear time framework:
(M,ϕ_{START},Φ) $\models$ p iff $\forall$x in M such that M,x $\models$ ϕ_{START} and M,x $\models$ Φ, we have M,x $\models$ p

(ii) in the branching time framework:
(M,ϕ_{START},Φ) $\models$ p iff $\forall$s in M such that M,s $\models$ ϕ_{START} we have M,s $\models$ p_Φ,
where p_Φ is the branching time formula obtained from p by relativizing all path quantification to scheduling constraint Φ; i.e., by replacing (starting at the innermost subformulae and working outward) each subformula Aq by A($\Phi \Rightarrow$ q) and Eq by E($\Phi \wedge$ q).

6 Theoretical Aspects of Temporal Logic

In this section we discuss the work that has been done in the Computing Science community on the more purely theoretical aspects of Temporal Logic. This work has tended to focus on decidability, complexity, axiomitizability, and expressiveness issues. Decidability and complexity refer to natural decision problems associated with a system of Temporal Logic including (i) satisfiability—given a formula, does there exist a structure that is a model of the formula?, (ii) validity—given a formula, is it true that every structure is a model of the formula?, and (iii) model checking—given a formula together with a particular finite structure, is the structure a model of the formula? (Note: a formula is valid iff its negation is not satisfiable, so satisfiability and validity are, in effect, equivalent problems.) Axiomitizability refers to the question of the existence of deductive systems for proving all the valid formulae of a system of Temporal Logic, and the investigation of their soundness and completeness properties. Expressiveness concerns what correctness properties can and cannot be formulated in a given logic. The bulk of theoretical work has thus been to analyze, classify, and compare various systems of Temporal Logic with respect to these criteria, and to study the tradeoffs between them. We remark that these issues are not only of intrinsic interest, but are also significant due to their implications for mechanical reasoning applications.

6.1 Expressiveness

6.1.1 Linear Time Expressiveness

It turns out that PLTL has intimate connections with formal language theory. This connection was first articulated in the literature by Wolper who argued in [Wo83] that PLTL "is not sufficiently expressive":

Theorem 6.1. The property G_2Q, meaning that "at all even times (0,2,4,6,...etc.), Q is true," is not expressible in PLTL.

To remedy this shortcoming Wolper [Wo83] suggested the use of an extended logic based on grammar operators; for example, the grammar

$$V_0 \rightarrow Q;\ \mathit{true};\ V_0$$

defines the set of models of G_2Q. This relation with formal languages is discussed in more detail subsequently.

Quantified PLTL

Another way to extend PLTL is to allow quantification over atomic propositions (cf. [Wo82], [Si83]). The syntax of PLTL is augmented by the formation rule:

if p is a formula and Q is an atomic proposition occurring free in p,
then $\exists$Qp is a formula also.

The semantics of $\exists$Qp is given by

M,x $\models \exists$Qp iff there exists a linear structure $M' = (S,x,L')$ such that $M',i \models p$ where $M = (S,x,L)$ and L' differs from L in at most the truth value of Q.

The formula $\exists$Qp thus represents existential quantification over Q; since, under the interpretation M, Q may be viewed as defining an infinite sequence of truth values, one for every state s along x, this is a type of *2nd order* quantification. We use $\forall$Qp to abbreviate $\neg\exists Q\ \neg p$; of course, it means universal quantification over Q.

The extended temporal operator G_2Q can be defined in QPLTL:

$$G_2Q \equiv_i \exists Q'(Q' \wedge X\neg Q' \wedge G(Q' \Leftrightarrow XXQ') \wedge G(Q' \Rightarrow Q))$$

It can be shown that QPLTL coincides in expressive power with a number of formalisms from language theory including the just discussed grammar operators of [Wo83].

6.1.2 Monadic Theories of Linear Ordering

The First-Order Language of Linear Order (FOLLO) is that formal system corresponding to the "Right-Hand-Side" of the definitions of the basic temporal operators of PLTL. A formula of FOLLO is interpreted over a linear time structure (S,<); for our purposes, we as usual only consider (|N,<) or (I,<) where I is an initial segment of |N. The language of linear order is built from the following symbols:

P, Q, ...etc. denoting monadic (1 argument) predicate symbols (and intuitively corresponding to atomic propositions),

t, u, ...etc. denoting individual variables (and intuitively ranging over moments of time in |N), and

< — the distinguished *less than* symbol (representing the temporal ordering)

A linear time structure M=(S,x,L) is then defined just as for PLTL; note that L may be viewed as assigning to each monadic predicate symbol in AP the set of times at which it is true.

The formulae of FOLLO are those generated by the following rules:

LO0: If t,u are individual variables then $t < u$ is a formula

LO1: If P is a monadic predicate symbol and t is an individual variable, then P(t) is a formula

LO2: If p,q are formulae then so are $p \wedge q$, $p \vee q$, $\neg p$

LO3: If p is a formula and t is a free individual variable in p, then $\exists t(p)$ is a formula

The Second Order Language of Linear Order (SOLLO) is obtained by using the following additional rule:

LO4: If Q is a monadic predicate symbol not in AP that appears free in formula p then $\exists Qp$ is a formula

The semantics of SOLLO and FOLLO are defined in the obvious way. The results depicted below indicate how the expressive powers of the variants of PLTL and the Theories of Linear Ordering compare:

Theorem 6.2. As measured with respect to initial equivalence, the relative expressive power of these linear time formalisms is as depicted below:

PLTLF $\equiv_i$ PLTLB $\equiv_i$ FOLLO $<_i$ SOLLO $\equiv_i$ QPLTLB $\equiv_i$ QPLTLF.

For the sake of thoroughness, we include

Theorem 6.3. As measured with respect to global equivalence, the relative expressive power of these linear time formalisms is as depicted below, where any two logics not connected by a chain of $\equiv_g$'s and $<_g$'s are of incomparable expressive power:

$$\begin{tikzcd}
& \text{PLTLB} \equiv_g \text{FOLLO} \arrow[dr, dash] & \\
\text{PLTLF} <_g \arrow[ur, dash] \arrow[dr, dash] & & <_g \text{SOLLO} \equiv_g \text{QPLTLB} \\
& \text{QPLTLF} \arrow[ur, dash] &
\end{tikzcd}$$

Note that QPLTLB (respectively, QPLTLF) denotes the version of PLTL with quantification over auxiliary propositions having both past and future tense temporal operators (respectively, future tense temporal operators only).

6.1.3 Regular Languages and PLTL

There is an intimate relationship between languages definable in (variants and extensions of) PLTL, the monadic theories of linear ordering, and the regular languages. We will first consider languages of finite strings, and then languages of infinite strings. In the sequel let Σ be a finite alphabet. For simplicity, we further assume Σ = PowerSet(AP) for some set of atomic propositions AP. Moreover, we assume that the empty string λ is excluded so that the languages of finite strings are subsets of Σ^+, rather than Σ^*.

Languages of Finite Strings

Before presenting the results we briefly review regular expression notations and certain concept concerning finite state automata. The reader is referred to [Th89] for more details.

There are several types of regular expression notations:

- the *restricted regular expressions* which are those built up from the alphabet symbols σ, for each $\sigma \in \delta$, and $\bullet$, $\cup$, and $*$, denoting "concatenation," "union," and "kleene (or star) closure" respectively.
- the *general regular expressions* which are those built up from the alphabet symbols σ for each $\sigma \in \Sigma$ and $\bullet$, $\cap$, $\neg$, $\cup$, $*$ denoting "concatenation," "intersection," "complementation (with respect to Σ^*)," "union," and "kleene (or star) closure" respectively.
- the *star-free regular expressions* are those general regular expressions with no occurrence of $*$.

The restricted regular expressions are equivalent in expressive power to the general regular expressions; however, the star-free regular expressions are strictly less expressive.

A finite state automaton M=(Q,Σ,δ,q_0,F) is said to be *counter-free* iff it is *not* the case that there exist distinct states $q_0,\ldots,q_{k-1} \in Q$, $k \geq 1$, and a word $w \in \Sigma^+$ such that $q_{i+1 \bmod k} \in \delta(q_i,w)$. A language L is said to be *noncounting* iff it is accepted by some counter-free finite state automaton. Intuitively, a counter-free automaton cannot count modulo n for any $n \geq 2$. It is also known that the noncounting languages coincide with those expressible by star-free regular expressions. We now have the following results:

Theorem 6.4. The following are equivalent conditions on languages L of finite strings:

(a) $L \subseteq \Sigma^+$ is definable in PLTL

(b) $L \subseteq \Sigma^+$ is definable in FOLLO

(c) $L \subseteq \Sigma^+$ is definable by a star-free regular expression

(d) $L \subseteq \Sigma^+$ is definable by a counter-free finite state automaton

This result thus accounts for why Wolper's property G_2P is not expressible in PLTL, for it requires counting modulo 2. The following result also suggests why his regular grammar operators suffice:

Theorem 6.5. The following are equivalent conditions for languages L of finite strings:

(a) $L \subseteq \Sigma^+$ is definable in QPLTL

(b) $L \subseteq \Sigma^+$ is definable in SOLLO

(c) $L \subseteq \Sigma^+$ is definable by a regular expression

(d) $L \subseteq \Sigma^+$ is definable by a finite state automaton

The equivalence of conditions (b), (c), and (d) was established using lengthy and difficult arguments in the monograph of McNaughton & Pappert [MP62]. The equivalence of conditions (a) and (b) in Theorem 6.4 was established in Kamp [Ka68], while for Theorem 6.5 it was established in [LPZ85]. Direct translations between PLTL and star-free regular expressions were given in [Zu86].

Remark: Since we have past tense operators, it is natural to think of history variables. If x = $(s_0,s_1,s_2,\ldots)$ is a computation then the most general history variable h would be that which at time j has accumulated the complete history $s_0 \ldots s_j$ up to (and including) time j. The expressive power of a language with history variables depends on the type of predicates we may apply to the history variables. One natural type of history predicate is of the form $[\alpha]_H$ where α is a (star-free) regular expression, with semantics given by

$(x,i) \models [\alpha]_H$ iff $s_0 \ldots s_i$ considered as a string over Σ = PowerSet(AP)
is in the language over Σ denoted by α

These history variables will be helpful in describing canonical forms for languages of infinite strings subsequently.

Languages of Infinite Strings

In extending the notion of regular language to encompass languages of infinite strings, the principle concern is how to finitely describe an infinite string. For finite state automata this is done using an extended notion of acceptance involving repeating a designated set of states infinitely often. See [Th89]. The framework of regular expressions can be similarly extended, in one of two ways:

(i) by adding an infinite repetition operator: ω. If α is an (ordinary) regular expression, then α^ω represents all strings of the form $a_1\ a_2\ a_3 \ldots$, where each $a_i \in \alpha$.

(ii) by adding a limit operator: lim. If α is an ordinary regular expression, then lim α consists of all those strings in Σ^ω which have infinitely many (distinct) finite prefixes in α.

We now have the two results below which follow from an assemblage of results in the literature (cf. [Ka68], [LPZ85], [Th89]):

Theorem 6.6 The following are equivalent conditions for languages L of infinite strings:

(a) $L \subseteq \Sigma^\omega$ is definable in QPLTL

(b) $L \subseteq \Sigma^\omega$ is definable in SOLLO

(c_0) $L \subseteq \Sigma^\omega$ is definable by an ω-regular expression, i.e. an expression of the form $\cup_{i=1}^{m} \alpha_i \beta_i^\omega$ where α_i, β_i are regular expressions

(c_1) $L \subseteq \Sigma^\omega$ is definable by an ω-limit regular expression, i.e. an expression of the form $\cup_{i=1}^{m} \alpha_i$ lim β_i where α_i, β_i are regular expressions

(c_2) $L \subseteq \Sigma^\omega$ is representable as $\cup_{i=1}^{m}$ (lim $\alpha_i \cap \neg$lim β_i) where α_i, β_i are regular expressions

(c_3) $L \subseteq \Sigma^\omega$ is expressible as $\vee_{i=1}^{m} (\overset{\infty}{F} [\alpha_i]_H \wedge \neg \overset{\infty}{F} [\beta_i]_H)$ where α_i, β_i are regular expressions

For the case of the star-free ω-languages we have

Theorem 6.7. The following are equivalent conditions for languages L of infinite strings:

(a) $L \subseteq \Sigma^\omega$ is definable in PLTL

(b) $L \subseteq \Sigma^\omega$ is definable in FOLLO

(c_1) $L \subseteq \Sigma^\omega$ is definable by an ω-regular expression, i.e. an expression of the form $\cup_{i=1}^{m} \alpha_i$ lim β_i where α_i, β_i are star-free regular expressions

(c_2) $L \subseteq \Sigma^\omega$ is representable as $\cup_{i=1}^{m}$ (lim $\alpha_i \cap \neg$lim β_i) where α_i, β_i are star-free regular expressions

(c_3) $L \subseteq \Sigma^\omega$ is expressible in the form $\vee_{i=1}^{m} (\overset{\infty}{F} [\alpha_i]_H \wedge \neg \overset{\infty}{F} [\beta_i]_H)$ where α_i, β_i are star-free regular expressions.

Result 6.7 (c_0) analogous to Result 6.6 (c_0) was intentionally omitted—because it does not hold as noted in [Th79]. It is not the case that $\cup_{i=1}^{m} \alpha_i \beta_{i=1}^{\omega}$ where α_i, β_i are star-free regular expressions, must itself denote a star-free regular set. For example, consider the language $L = (00 \cup 1)^\omega$. L is expressible as a union of $\alpha_i \beta_i^\omega$: take $m = 1$, $\alpha_i = \epsilon$, $\beta_1 = 00 \cup 1$. But L, which consists intuitively of exactly those strings for which there is an even number of 0's between every consecutive pair of 1's, is not definable in FOLLO, nor is it star-free regular.

Remark: One significant issue we do not address here in any detail — and which is not very thoroughly studied in the literature — is that of *succinctness*. Here we refer to how long or short a formula is needed to capture a given correctness property. Two formalisms may have the same raw expressive power, but one may be much more succinct than the other. For example, while FOLLO and PLTL have the same raw expressive power, it is known that FOLLO can be significantly (nonelementarily) more succinct than PLTL (cf. [Me74]).

6.1.4 Branching Time Expressiveness

Analogy with the linear temporal framework suggests several formalisms for describing infinite trees that might be compared with branching temporal logic. Among these are: finite state automata on infinite trees, the monadic second order theory of many successors (SnS), and the monadic second order theory of partial orders. However, not nearly so much is known about the comparison with related formalisms in the branching time case.

One difficulty is that, technically, the types of branching objects considered differ. Branching Temporal Logic is interpreted over structures which are, in effect, trees with nodes of infinite outdegree, whereas, e.g., tree automata take input trees of fixed finite outdegree. Another difficulty is that the logics, such as CTL*, as ordinarily considered, do not distinguish between, e.g., left and right successor nodes, whereas the tree automata can.

To facilitate a technical comparison, we therefore restrict our attention to (a) structures corresponding to infinite binary trees and (b) tree automata with a "symmetric" transition function that do not distinguish, e.g., left from right. Then we have the following result from [ESi84] comparing logics augmented with existential quantification over atomic propositions with tree automata.

Theorem 6.8.

(i) EQCTL* is exactly as expressive as symmetric pairs automata on infinite trees

(ii) EQCTL is exactly as expressive as symmetric Buchi automaton infinite trees.

Here, EQCTL* consists of the formula $\exists Q_1 \ldots \exists Q_m f$, where f is a CTL* formula and the Q_i are atomic propositions appearing in f. The semantics is that, given a structure M=(S,R,L), $M,s \models \exists Q_1 \ldots \exists Q_m f$ iff there exists a structure $M'=(S,R,L')$ such that $M',s \models f$ and L' differs from L at most in the truth assignments to each Q_i, $1 \leq i \leq m$. Similarly, EQCTL consists of formulae $\exists Q_1 \ldots Q_m f$, where f is a CTL formula.

A related result is from [HT87]:

Theorem 6.9. CTL* is exactly as expressive as the monadic second order theory of two successors with set quantification restricted to infinite paths, over infinite binary trees.

Remark: By augmenting CTL* with arc assertions which allow it to distinguish outgoing arc i from arc i+1 the result extends to infinite n-ary trees, $n > 2$. By taking $n = 1$, the result specializes to the "expressive completeness" result of Kamp [Ka68] that PLTL is equivalent in expressive power to FOLLO (our Theorem 6.7 (a,b)).

While less is known about comparisons of BTLs (Branching Time Logics) against external "yardsticks," a great deal is known about comparisons of BTLs against each other. This contrasts with the reversed situation for LTLs (Linear Time Logics). Perhaps this reflects the much greater degree of "freedom" due to the multiplicity of alternative futures found in the BTL framework.

It is useful to define the notion of a *basic modality* of a BTL. This is a formula of the form Ap or the form Ep, where p is a pure linear time formula (containing no path quantifiers.) Then a formula of a logic may be seen as being built up by combining basic modalities using boolean connectives and nesting. For example, EFP is a CTL basic modality; so is AFQ. EFAFQ is formula of CTL (but not a basic modality) obtained by nesting AFQ within EFP (more precisely, by substituting AFQ for P within EFP). E(FP $\wedge$ FQ) is a basic modality of CTL*, but not a basic modality nor a formula of CTL.

A large number of sublanguages of CTL* can be defined by controlling the way the linear time operators combine using boolean connectives and nesting of operators in the basic modalities of the language. For instance, we use B(F,X,U) to indicate the language where only a single linear time operator X, F, or U can follow a path quantifier, and B(F,X,U,$\wedge$,$\neg$) to indicate the language where boolean combinations of these linear operators are allowed, but not nesting of the linear operators. Thus formula E(Fp $\wedge$ Gq) is in the language B(F,X,U,$\wedge$,$\neg$) but not in B(X,F,U).

The diagram in Figure 3 shows how some of these logics compare in expressive power. The notation $L_1 < L_2$ means that L_1 is strictly less expressive than L_2, which holds provided that

(a) $\forall$ formula p of L_1 $\exists$ a formula q of L_2 such that $\forall$ structure M $\forall$ state s in M, M,s $\models$ p iff M,s $\models$ q, and

(b) the converse of (a) does not hold,

while $L_1 \equiv L_2$ means L_1 and L_2 are equivalent in expressive power, and $L_1 \leq L_2$ means $L_1 < L_2$ or $L_1 \equiv L_2$.

Most of the logics shown are known from the literature. B(F) is the branching time logic of Lamport [La80], having basic modalities of the form A or E followed by F or G. The logic B(X,F), which has basic modalities of the form A or E followed by X, F, or G, was originally proposed in [BPM82] as the logic UB. The logic B(X,F,U) is of course CTL. The logic B(X,F,U,$\overset{\infty}{F}$,$\wedge$,$\neg$) is essentially the logic proposed in [EC80]; its infinitary modalities $\overset{\infty}{F}$ and $\overset{\infty}{G}$ permit specification of fairness properties.

We now give some rough, high-level intuition underlying these results. Semantic containment along each edge follows directly from syntactic containment in all cases, except edges 2 and 4, which follow given the semantic equivalence of edge 3 (discussed below).

The X operator (obviously) cannot be expressed in terms of the F operator, which accounts for edge 0, B(F) $\leq$ B(F,X). Similarly, the U operator cannot be expressed in terms of X, F, and boolean connectives. This was known “classically” (cf. [Ka68]), and accounts for edge 2: B(X,F,$\wedge$,$\neg$) < B(X,F,U).

To establish the equivalence of edge 3, we need to provide a translation of B(X,F,U,$\wedge$,$\neg$) into B(X,F,U). The basic idea behind this translation can be understood by noting that E(FP $\wedge$ FQ) $\equiv$ EF(P $\wedge$ EFQ) $\vee$ EF(Q $\wedge$ EFP). However, it is a bit more subtle than that; the ability to do the translation in all cases depends on the presence of the until (U) operator (cf. edge 1). The following validities, two of which concern the until, can be used to inductively translate each B(X,F,U,$\wedge$,$\neg$) formula into an equivalent B(X,F,U) formula:

$$E(\ (p_1\ U\ q_1) \wedge (p_2\ U\ q_2)\ ) \equiv$$
$$E(\ (p_1 \wedge p_2)\ U\ (q_1 \wedge E(p_2\ U\ q_2))) \vee E(\ (p_2 \wedge p_1)\ U\ (q_2 \wedge E(p_1\ U\ q_1)))$$
$$E(\neg(p\ U\ q)) \equiv E(\ (\neg q \wedge \neg p)\ U\ (q \wedge p)\ ) \vee EG\neg q$$
$$E(\neg Xp) \equiv EX\neg p$$

$E\overset{\infty}{F}Q$, a $B(X,F,U,\overset{\infty}{F})$ formula is not expressible in $B(X,F,U)$ accounting for the strict containment on arc 4. This is probably the most significant result, for it basically says that correctness under fairness assumptions cannot be expressed in a BTL with a simple set of modalities. For example, the property that P eventually becomes true along all fair computations (fair inevitability of P) is of the form $A(\overset{\infty}{F}Q \Rightarrow FP)$ for even a (very) simple fairness constraint like $\overset{\infty}{F}Q$. Neither it, nor its dual $E(\overset{\infty}{F}Q \wedge GP)$, is expressible in $B(X,F,U)$, since by taking P to be *true* the dual becomes $E\overset{\infty}{F}Q$.

The inexpressibility of $E\overset{\infty}{F}Q$ was established in [EC80], using recursion-theoretic arguments to show that the predicate transformer associated with $E\overset{\infty}{F}Q$ is Σ^1_1-complete while the predicate transformers for $B(X,F,U)$ are arithmetical. The underlying intuition is that $E\overset{\infty}{F}Q$ uses second order quantification in an essential way to assert that that there exists a sequence of nodes in the computation tree where Q holds. Another version of this inexpressiveness result was established by Lamport [La80] in a somewhat different technical framework. Still another proof of this result was given by Emerson and Halpern [EH86]. The type of inductive, combinatorial proof used is paradigmatic of the proofs of many inexpressiveness results for TL, so we describe the main idea here.

Theorem 6.10. $E\overset{\infty}{F}Q$ is not expressible in $B(X,F,U)$

Proof Idea. We inductively define two sequences $M_1,M_2,M_3,\ldots$ and $N_1,N_2,N_3,\ldots$ of structures as shown in Figure 4. It is plain that for all i,

(*) $M_i,s_i \models E\overset{\infty}{F}Q$ and $N_i,s_i \models \neg E\overset{\infty}{F}Q$

Thus $E\overset{\infty}{F}Q$ distinguishes between the two sequences. However, we can show by an inductive argument that each formula of $B(X,F,U)$ is "confused" by the two sequences, in that

(**) $M_i,s_i \models p$ iff $N_i,s_i \models p$

If some formula p of $B(X,F,U)$ were equivalent to $E\overset{\infty}{F}Q$, we would then have for $i =$ the length of p that

$M,s_i \models p$ and $N,s_i \models \neg p$ by virtue of (*)
and also that
$N,s_i \models p$, by virtue of (**), a contradiction. □

The strict containment along the rest of the edges follow from these inexpressiveness results: $E(FP \wedge GQ)$ is not expressible in $B(X,F)$, for edge 1. $E(\overset{\infty}{F}P_1 \wedge \overset{\infty}{F}P_2)$ is not expressible in $B(X,F,U,\overset{\infty}{F})$, for edge 5. $A(F(P \wedge XP))$ is not expressible in $B(X,F,U,\overset{\infty}{F},\wedge,\neg)$, for edge 6. The proofs are along the lines of the theorem above for $E\overset{\infty}{F}Q$.

It is also possible to compare branching with linear time logics. When a linear time formula is interpreted over a program, there is usually an implicit universal quantification over all possible computations. This suggests that when given a linear time language L, which is of course a set of path formulae, we convert it into a branching time language by prefixing each path formula by the universal path quantifier A. We thus get the corresponding branching language BL(L) = { Ap: p $\in$ L}. Figure 5 shows how various branching and linear logics compare. Not surprisingly, the major limitation of linear time is its inability to express existential path quantification (cf. [La80], [EH86]).

Theorem 6.11. The formula EFP is not expressible in any BL(—) logic.

6.2 Decision Procedures for Propositional Temporal Logics

In this section we discuss algorithms for testing if a given formula p_0 in a system of propositional TL is satisfiable. The usual approach to developing such algorithms is to first establish the *small model property* for the logic: if a formula is satisfiable, then it is satisfiable in a "small" finite model, where "small" means of size bounded by some function, say, f, of the length of the input formula. This immediately yields a decision procedure for the logic. Guess a small structure M as a candidate model of given formula p_0; then check that M is indeed a model of p_0. This check can be done by exhaustive search, since M is finite, and can often be done efficiently.

An elegant technique for establishing the small model property is through use of the *quotient construction*, also called—in classical model logic—*filtration*, where an equivalence relation of small finite index is defined on states. Then equivalent states are identified to collapse a possibly infinite model to a small finite one.

An example of a quotient construction is its application to yield a decision procedure for Propositional Dynamic Logic of [FL79], discussed in [KT89]. There the equivalence relation is defined so that, in essence, two states are equivalent when they agree (i.e., have the same truth value) on all subformulae of the formula p_0 being tested for satisfiability. This yields a decision procedure of nondeterministic exponential time complexity, calculated as follows. The total complexity is the time to guess a small candidate model plus the time to check that it is indeed a model. The candidate model can be guessed in time polynomial in its size which is exponential in the length of p_0, since for a formula of length n there are about n subformulae and 2^n equivalence classes. And it turns out that checking that the candidate model is a genuine model can be done in polynomial time.

Of course the deterministic time complexity of the above algorithm is double exponential. The complexity can be improved through use of the tableau construction.

A *tableau* for formula p_0 is a finite directed graph with nodes labelled by subformulae associated with p_0 that, in effect, encodes all potential models of p_0. In particular, as in the case of Propositional Dynamic Logic, the tableau contains as a subgraph the quotient structure corresponding to any model of p_0. The tableau can be constructed, and then tested for consistency to see if it contains a genuine quotient model. Such testing can often be done efficiently. In the case of of Propositional Dynamic Logic, the tableau is of size exponential in the formula length, while the testing can be done in deterministic polynomial time in the tableau size, yielding a deterministic single exponential time decision procedure.

For some logics, no matter how we define a finite index equivalence relation on states, the quotient construction yields a quotient structure that is not a model. However, for many logics, the quotient structure still provides useful information. It can be viewed as a "pseudo-model" that can be unwound into a genuine, yet still small, model. The tableau construction, moreover, can still be used to perform a systematic search for a pseudo-model, to be unwound into a genuine model.

We remark that the tableau construction is a rather general one, that applies to many logics. Tableau-based decision procedures for various logics are given in [Pr79], [BPM81], [BHP82], [Wo82], [Wo83], [HS84]. See also the excellent survey by Wolper [Wo84]. In the sequel we describe a tableau-based decision procedure for CTL formulae, along the lines of [EC82] and [EH85]. The following definitions and terminology are needed.

We assume that the candidate formula p_0 is in *positive normal form*, obtained by pushing negations inward as far as possible using de Morgan's laws ($\neg(p \vee q) \equiv \neg p \wedge \neg q$, $\neg(p \wedge q) \equiv \neg p \vee \neg q$) and dualities ($\neg AGp \equiv EF\neg p$, $\neg A[p\ U\ q] \equiv E[\neg p\ B\ q]$, etc.). This at most doubles the length of the formula, and results in only atomic propositions being negated. We write $\sim p$ for the formula in positive normal form equivalent to $\neg p$. The *closure* of p_0, $cl(p_0)$, is the least set of subformulae such that:

- Each subformulae of p_0, including p_0 itself, is a member of $cl(p_0)$;
- If EFq, EGq, E[p U q], or E[p B q] $\in cl(p_0)$ then, respectively, EXEFq, EXEGq, EXE[p U q], or EXE[p B q] $\in cl(p_0)$;
- If AFq, AGq, A[p U q], or A[p B q] $\in cl(p_0)$ then, respectively, AXAFq, AXAGq, AXA[p U q], or AXA[p B q] $\in cl(p_0)$;

The *extended closure* of p_0, $ecl(p_0) = cl(p_0) \cup \{\sim p\colon\ p \in cl(p_0)\}$. Note that $card(ecl(p_0)) = O(length(p_0))$.

At this point we give the technical definitions for the quotient construction, as they are needed in the proof of the small model theorem of CTL. We also show the the quotient construction by itself is inadequate for getting a small model theorem for CTL.

Let M=(S,R,L) be a model of p_0, let H be a set of formulae, and let $\equiv_H$ be an equivalence relation on S induced by agreement on the formulae in H, i.e. $s \equiv_H t$ whenever $\forall\, q \in H$, $M,s \models q$ iff $M,t \models q$. We use [s] to denote the equivalence class $\{t\colon\ t \equiv_H s\}$ of s. Then the quotient structure of M by $\equiv_h$, $M/\!\equiv_H$, $= (S',R',L')$ where $S' = \{[s]\colon\ s \in S\}$, $R' = \{([s],[t])\colon\ (s,t) \in R\}$, and $L'([s]) = L(s) \cap H$. Ordinarily, we take $H = ecl(p_0)$.

However, as the following theorem shows, no way of defining the equivalence relation for the quotient construction preserves modelhood:

Theorem 6.12. For every set H of (CTL) formulae, the quotient construction does not preserve modelhood for the formula AFP. In particular, there is a model M of AFP such that for every finite set H, $M/\!\equiv_H$ is not a model for AFP.

Proof Idea. Note the structure shown in Figure 6(a) is a model of AFP. But however the quotient relation collapses the structure two distinct states s_i and s_j will be identified, resulting in a cycle in the quotient structure, along which P is always false, as suggested in Figure 6(b). Hence AFP does not hold along the cycle. □

We now proceed with the technical development needed. To simplify the exposition, we assume that the candidate formula p_0 is of the form $p_1 \wedge$ AGEX*true*, syntactically reflecting the semantic requirement that each state in a structure have a successor state.

We say that a formula is *elementary* provided that it is a proposition, the negation of a proposition, or has main connective AX or EX. Any other formula is *nonelementary*. Each nonelementary formula may be viewed as either a conjunctive formula $\alpha \equiv \alpha_1 \wedge \alpha_2$ or a disjunctive formula $\beta \equiv \beta_1 \vee \beta_2$. Clearly, $f \wedge g$ is an α formula and $f \vee g$ is a β formula. A modal formula may be classified as α or β based on its fixpoint characterization (cf. section 8.4); e.g., EFp = p $\vee$ EXEFp is a β formula and AGp = p $\wedge$ AXAGp is an α formula. The following table summarizes the classification:

α = p $\wedge$ q	α_1 = p	α_2 = q
α = A[p B q]	α_1 = ∼q	α_2 = p $\vee$ AXA[p B q]
α = E[p B q]	α_1 = ∼q	α_2 = p $\vee$ EXE[p B q]
α = AGq	α_1 = q	α_2 = AXAGq
α = EGq	α_1 = q	α_2 = EXEGq
β = p $\vee$ q	β_1 = p	β_2 = q
β = A[p U q]	β_1 = q	β_2 = p $\wedge$ AXA[p U q]
β = E[p U q]	β_1 = q	β_2 = p $\wedge$ EXE[p U q]
β = AFq	β_1 = q	β_2 = AXAFq
β = EFq	β_1 = q	β_2 = EXEFq

A formula of the form A[p U q] or E[p U q] is an *eventuality* formula. An eventuality makes a promise that something will happen. This promise must be *fulfilled*. The eventuality A[p U q] (E[p U q]) is fulfilled for s in M provided that for every (respectively, for some) path starting at s, there exists a finite prefix of the path in M whose last state is labelled with q and all of whose other states are labelled with p. Since AFq and EFq are special cases of A[p U q] and E[p U q], respectively, they are also eventualities. In contrast, A[p B q], E[p B q], and their special cases AGq and EGq, are *invariance* formulae. An invariance property asserts that whatever happens to occur (if anything) will meet certain conditions (cf. subsection 7.1).

We say that a *prestructure* M is a triple (S,R,L) just like a structure except that the binary relation R is not required to be total. An *interior* node of a prestructure is one with at least one successor. A *frontier* node is one with no successors.

It is helpful to associate certain consistency requirements on the labelling of a (pre)structure:

Propositional Consistency Rules:

PC0 $\sim$p $\in$ L(s) implies p $\notin$ L(s)
PC1 $\alpha \in$ L(s) implies $\alpha_1 \in$ L(s) and $\alpha_2 \in$ L(s)
PC2 $\beta \in$ L(s) implies $\beta_1 \in$ L(s) or $\beta_2 \in$ L(s)

Local Consistency Rules:

LC0 AXp $\in$ L(s) implies $\forall$ successor t of s, p $\in$ L(t)
LC1 EXp $\in$ L(s) implies $\exists$ successor t of s, p $\in$ L(t)

A *fragment* is a prestructure whose graph is a dag (directed acyclic graph) such that all of its nodes satisfy PC0-2 and LC0 above, and all of its interior nodes satisfy LC1 above.

A *Hintikka structure (for p_0)* is a structure M=(S,R,L) (with $p_0 \in$ L(s) for some s $\in$ S) which meets the following conditions:

1. the propositional consistency rules PC0-2,
2. the local consistency rules LC0-1, and
3. each eventuality is fulfilled.

Proposition 6.13. If structure M = (S,R,L) defines a model of p_0 and each state s is labelled with exactly the formula in ecl(p_0) true at s, then M is a Hintikka structure for p_0. Conversely, a Hintikka structure for p_0 defines a model of p_0.

If M is a Hintikka structure, then for each node s of M and each eventuality r in ecl(p_0) such that M,s $\models$ r, there is a fragment, call it DAG[s,r], which certifies fulfillment of r at s in M. What is the nature of this fragment? It has s as its root, i.e., node from which all other nodes in DAG[s,r] are reachable. If r is of the form AFq, then DAG[s, AFq] is obtained by taking node s and all nodes along all paths emanating from s up to and including the first state where q is true. The resulting subgraph is indeed a dag all of whose frontier nodes are labelled with q. If r were of the form A[p U q], DAG[s, A[p U q]] would be the same except its interior nodes are all labelled with p. In the case of DAG[s, EFq] take a shortest path leading from node s to a node labelled with q, and then add sufficient successors to ensure that LC1 holds of each interior node on the path. In the case of DAG[s, E[p U q]], the only change is that p labels each interior node on the path.

In a Hintikka structure M for p_0, each fulfilling fragment DAG[s, r] for each eventuality r, is "cleanly embedded" in M. If we collapse M by applying a finite index quotient construction, the resulting quotient structure is not, in general, a model because cycles are introduced into such fragments. However, there is still a fragment, call it DAG$'$[s,r], "contained" in the quotient structure of M. It is simply no longer cleanly embedded. Technically, we say prestructure M_1 = (S_1,R_1,L_1) is *contained* in prestructure M_2 = (S_2,R_2,L_2) whenever $S_1 \subseteq S_2$, $R_1 \subseteq R_2$, and $L_1 = L_2|S_1$, the labelling L_2 restricted to S_1. We say M_1 is *cleanly embedded* in M_2 provided M_1 is contained in M_2, and also every interior node of M_1 has the same set of successors in M_1 as in M_2.

A *pseudo-Hintikka structure (for p_0)* is a structure M=(S,R,L) (with $p_0 \in$ L(s) for some s $\in$ S) which meets the following conditions:

1. the propositional consistency rules PC0-2,

2. the local consistency rules LC0-1, and

3. each eventuality is *pseudo-fulfilled* in the following sense:

AFq $\in$ L(s) (resp., A[p U q] $\in$ L(s))
implies there is a finite fragment—called DAG[s, AFq] (resp., DAG[s, A[p U q]])—
rooted at s contained in M such that
for *all* frontier nodes t of the fragment, q $\in$ L(t)
(resp., and for all interior nodes u of the fragment, p $\in$ L(u));

EFq $\in$ L(s) (resp., E[p U q] $\in$ L(s))
implies there is a finite fragment—called DAG[s, EFq] (resp., DAG[s, E[p U q]])—
rooted at s contained in M such that
for *some* frontier node t of the fragment, q $\in$ L(t)
(resp., and for all interior nodes u of the fragment, p $\in$ L(u)).

Theorem 6.14. (Small Model Theorem for CTL) Let p_0 be a CTL formula of length n. Then the following are equivalent:

(a) p_0 is satisfiable
(b) p_0 has a infinite tree model with finite branching bounded by O(n)
(c) p_0 has a finite model of size $\leq$ exp(n)
(d) p_0 has a finite pseudo-Hintikka structure of size $\leq$ exp(n)

Proof Sketch: We show that (a) $\Rightarrow$ (b) $\Rightarrow$ (d) $\Rightarrow$ (c) $\Rightarrow$ (a).

(a) $\Rightarrow$ (b): Suppose M,s $\models p_0$. Then as described in subsection 5.1. M can be unwound into an infinite tree model M_1, with root state s_1 a copy of s. It is possible that M_1 has infinite branching at some states, so (if needed) we chop out spurious successor states to get a bounded branching subtree M_2 of M_1 such that still $M_2,s_1 \models p_0$. We proceed down M_1 level-by-level deleting all but n successors of each state. The key idea is that for each formula EXq $\in$ L(s), where s is a retained node on the current level, we keep a successor t of s of least q-rank, where the q-rank(s) is defined as the length of the shortest path from s fulfilling q, if q is of the form EFr or E[p U r], and is defined as 0 if q is of any other form. This will ensure that each eventuality of the form EFr or E[p U r] is fulfilled in the tree model M_2. Moreover, since there are at most O(n) formulae of the form EXq in ecl(p_0), the branching at each state of M_2 is bounded by O(n).

(b) $\Rightarrow$ (d): Let M be a bounded branching infinite tree model with root s_0, such that M,$s_0 \models p_0$. We claim that the quotient structure $M' = M/\equiv_{ecl(p_0)}$ is a pseudo-Hintikka structure. It suffices to show that for each state [s] of M', and each eventuality r in the label of [s] there is a finite fragment contained in M' certifying pseudo-fulfillment of r. We sketch the argument in the case r = AFq. The argument for other types of eventuality is similar.

So suppose AFq appears in the label of [s]. By definition of the quotient construction, in the original structure M AFq is true at state s, and thus there exists a finite fragment DAG[s, AFq] with root s cleanly embedded in M. Extract (a copy of) the fragment DAG[s, AFq]. Chop out states with duplicate labels. Given two states s, s' with the same label, let the deeper state replace the shallower, where the depth of a state is the length of the longest path from the state back to the root s_0. This ensures that after the more shallow node has been chopped out, the resulting graph is still a dag, and moreover, a fragment. Since we can chop out any pair of duplicates the final fragment, call it DAG′[[s],AFq] has at most a single occurrence of each label. Therefore (a copy of) DAG′[[s], AFq] is contained in the quotient structure M′. It follows that M′ is a pseudo-Hintikka model as desired.

(d) $\Rightarrow$ (c): Let M = (S,R,L) be a pseudo-Hintikka model for p_0. For simplicity we identify a state s with its label L(s). Then for each state s and each eventuality q $\in$ s, there is a fragment DAG[s,q] contained in M certifying fulfillment of q. We show how to splice together copies of the DAGs, in effect unwinding M, to obtain a Hintikka model for p_0.

For each state s and each eventuality q, we construct a dag rooted at s, DAGG[s,q]. If q $\in$ s then DAGG[s,q] = DAG[s,q]; otherwise DAGG[s,q] is taken to be the subgraph consisting of s plus a sufficient set of successors to ensure that local consistency rules LC0-1 are met.

We now take (a single copy of) each DAGG[s,q] and arrange them in a matrix as shown in Figure 7, the rows range over eventualities $q_1,\ldots q_m$ and the columns range over the states $s_1,\ldots,s_N$ in the tableau. Now each frontier node s in row i is replaced by the copy of s that is the root of DAGG[s,q_{i+1}] in row i+1. Note that each fullpath through the resulting structure goes through each row infinitely often. As a consequence, the resulting graph defines a model of p_0, as can be verified by induction on the structure of formulae. The essential point is that each eventuality q_i is fulfilled along each fullpath where needed, at least by the time the fullpath has gone through row i.

The cyclic model consists of m·N DAGG's, each consisting of N nodes. It is thus of size $m \cdot N^2$ nodes, where the number of eventualities $m \leq n$ and the number of tableau nodes $N \leq 2^n$, and n is the length of p_0. We can chop out duplicate nodes with the same label within a row, using an argument based on the depth of a node like that used above in the proof of (b) $\Rightarrow$ (d), to get a model of size m·N = exp(n).

(c) $\Rightarrow$ (a) is immediate. □

We now describe the tableau-based decision procedure for CTL. Let p_0 be the candidate CTL formula which is to be tested for satisfiability. We proceed as follows.

1. Build an initial *tableau* T=(S,R,L) for p_0, which encodes potential pseudo-Hintikka structures for p_0. Let S be the collection of all maximal, propositionally consistent subsets s of ecl(p_0), where by maximal we mean that for every formula p $\in$ ecl(p_0), either p or ∼p $\in$ s, while propositionally consistent refers to rules PC0-2 above. Let R $\subseteq$ S · S be defined so that (s,t) $\in$ R unless AXp $\in$ s and not(p) $\in$ t, for some formula AXp $\in$ ecl(p_0). Let L(s) = s. Note that the tableau as initially constructed meets all propositional consistency rules PC0-2 and local consistency rule LC0.

2. Test the tableau for consistency and pseudo-fulfillment of eventualities, by repeatedly applying the following deletion rules until no more nodes in the tableau can be deleted:

   - Delete any state s such that eventuality r $\in$ L(s) and there does *not* exist a fragment DAG[s,r] rooted at s contained in the tableau which certifies pseudo-fulfillment of r.
   - Delete any state which has no successors.
   - Delete any state which violates LC1.

Note that this portion of the algorithm must terminate, since there are only a finite number of nodes in the tableau.

3. Let T′ be the final tableau. If there exists a state s′ in T′ with $p_0 \in L(s')$ then return "YES, p_0 is satisfiable."; If not, then return "NO, p_0 is unsatisfiable".

To test the tableau for the existence of the appropriate fragments to certify fulfillment of eventualities we can use a ranking procedure. For an A[p U q] eventuality initially assign rank 1 to all nodes labelled with q and rank ∞ to all other nodes. Then for each node s and each formula r such that EXr is in the label of s, define $SUCC_r(s) = \{s''$: s' is a successor of s in the tableau with $r \in$ label of $s''\}$ and compute $rank(SUCC_r(s)) = \min\ \{rank\ s'$: $s'' \in SUCC_r(s)\}$. Now for each node s of rank $= \infty$ such that $p \in L(s)$ let $rank(s) = 1 + \max\ \{rank(SUCC_r(s)$: $EXr \in L(s)\}$. Repeatedly apply the above ranking rules until stabilization. A node has finite rank iff A[p U q] is fulfilled at it in the tableau. To test for fulfillment of an AFq is a special case of the above, ignoring the formula p. To test for fulfillment of E[p U q] use a procedure like the above, but compute rank(s) $= 1 + \min\ \{rank(SUCC_r(s)$: $EXr \in L(s)\}$. To test for fulfillment of EFq is again a special case, where the formula p is ignored.

Theorem 6.15. The problem of testing satisfiability for CTL is complete for deterministic exponential time.

Proof idea. The above algorithm can be shown to run in deterministic exponential time in the length of the input formula, since the size of the tableau is, in general, exponential in the formula size, and the tableau can be constructed and tested for containment of a pseudo-Hintikka structure in time polynomial in it size. This establishes the upper bound. The lower bound follows by a reduction from alternating polynomial space bounded Turing machines, similar to that used to establish exponential time hardness for Propositional Dynamic Logic (see [KT89]). □

The above formulation of the CTL decision procedure is sometimes known as the *maximal model* approach, since the nodes in the initial tableau are maximal, propositionally consistent sets of formulae and we put in as many arcs as possible. One drawback is that its average case complexity is as bad as its worst case complexity, since it always constructs the exponential size collection of maximal, propositionally consistent sets of formulae.

An alternative approach is to build the initial tableau incrementally, which in practice often results in a significant decrease in its size and time required to construct it. The tableau construction will now begin with a bipartite graph $T' = (C,D,R_{CD},R_{DC},L)$ where nodes in C are referred to as states while nodes in D are known as prestates; $R_{CD} \subseteq C \times D$ and $R_{DC} \subseteq D \times C$. The labels of the states will be *sparsely downward closed* sets of formulae in $ecl(p_0)$, i.e., sets which satisfy PC0, PC1, and PC2′: $\beta \in L(s)$ implies either $\beta_1 \in L(s)$ or $\beta_2 \in L(s)$.

Initially, let C = the empty set, D = a single prestate d labelled with p_0.

```
Repeat
    Let e be a frontier node of $T'$
    If e is a prestate d then
        let $c_1,\ldots,c_k$ be states whose labels comprise all the sparsely downward closed supersets of $L(d)$
        add $c_1,\ldots,c_k$ as $R_{DC}$-successors of d in $T'$
        Note: if any $c_i$ has the same label as another state $c'$ already
            in $T'$, then identify $c_i$ and $c'$ (i.e., delete $c_i$
            and draw an $R_{DC}$-arc from d to $c'$.)
    If e is a state c labelled with nexttime formulae $AXp_1,\ldots,AXp_l,EXq_1,\ldots,EXq_k$ then
        create prestates $d_1,\ldots,d_k$ labelled with sets resp. $\{p_1,\ldots,p_l,q_1\},\ldots,\{p_1,\ldots,p_l,q_k\}$
        and add them as $R_{CD}$-successors to c in $T'$
        Note: if any $d_i$ has the same label as another prestate $d'$ already
            in $T'$, then identify $d_i$ and $d'$ as above
Until all nodes in $T'$ have at least one successor
```

Now the tableau $T = (C,R,L|C)$ where C is the set of states in T' above and $R = R_{CD} \cdot R_{DC}$, $L|C$ is the labelling L restricted to C. Then the remainder of the decision procedure described previously can be applied to this new tableau constructed incrementally.

Remark: It is possible to construct the original type of tableau incrementally. Let the initial prestate be labelled with $p_0 \vee \sim p_0$ and use maximal, propositionally consistent sets for the labels of states.

The decision procedure for CTL also yields a deterministic exponential time decision procedure for PLTL:

Theorem 6.16. Let p_0 be a PLTL formula in positive normal form. Let p_1 be the CTL formula obtained from p_0 by replacing each temporal operator F, G, X, U, B by AF, AG, AX, AU, AB, resp. Then p_0 is satisfiable iff p_1 is satisfiable.

We can in fact do better for PLTL and various fragments of it. The following results on the complexity of deciding linear time are due to Sistla and Clarke [SC85]:

Theorem 6.17. The problem of testing satisfiability for PLTL is PSPACE-complete.

Proof Idea. To establish membership in PSPACE, we design a nondeterministic algorithm that, given an input formula p_0, guesses a satisfying path through the tableau for p_0 which defines a linear model of size exp(n), where $n = \text{length}(p_0)$. This path can be guessed and verified to be a model in only O(n) space, since the algorithm need only remember the label of the current and next state along the path, and the point where the path loops back, in order to check that eventualities are fulfilled. PSPACE-hardness can be established by a generic reduction from polynomial space Turing machines. □

For the sublanguage of PLTL restricted to allow only the F operator (and its dual G), denoted PLTL(F) further improvement is still possible. We first establish the somewhat surprising

Theorem 6.18. (Linear Size Model Theorem for PLTL(F)) If PLTL(F) formula p_0 of length n is satisfiable, then it has a finite linear model of size O(n).

Proof idea. The important insight is that truth of a PLTL(F) formula only depends on the set of successor states, and not their order or arrangement. Now suppose p_0 is satisfiable. Let x =

$s_0, s_1, s_2, \ldots$ be a model of p_0. Then there exist i and j such that $i < j$ and $s_i = s_j$ and the set of states appearing infinitely often along x equals $\{s_i, \ldots, s_{j-1}\}$. Let x' be the linear structure obtained be deleting all states of index greater than j-1 and making s_i the successor of s_{j-1}. It is readily checked that $x' \models p_0$. Moreover, since the order of successor states does not matter we can, in general, delete many states, while preserving the truth of p_0 in the resulting linear structure. We need only retain in the "loop," from state s_i to s_{j-1} and back, a single state labelled q, for each formula Fq that appears in the label some state in the loop. The other states in the loop can be deleted, reducing its size to at most n states. We also need to ensure that each Fq that appears somewhere in the "stem," from s_0 to s_{i-1}, is fulfilled by a q labelling some subsequent state. The other states in the stem can be deleted reducing the size of the stem to at most n states. The final structure, x'', is still a model of p_0, and is of size at most 2n states. □

Theorem 6.19. The problem of testing satisfiability for PLTL(F) is NP-complete.

Proof Idea. Membership in NP follows using the Linear Size Model Theorem. An algorithm can be designed that, given a formula of length n, guesses a candidate model of size O(n) and then checks that it is indeed a model in time $O(n^2)$. NP-hardness follows since the logic subsumes propositional logic. □

Finally, it can be shown that the complexity of testing satisfiability of the very expressive branching time logic CTL* has an upper bound of deterministic double exponential time, by means of a quite elaborate reduction to the nonemptiness problem for finite state automata on infinite trees (see section 6.5). A lower bound of deterministic double exponential time has also been established by a reduction from alternating exponential space Turing machines in [VS85]. (Note: By double exponential we mean exp(exp(n)), where exp(n) is a function c^n, for some $c > 1$.) Thus we have,

Theorem 6.20. The problem of testing satisfiability for CTL* is complete for deterministic double exponential time.

6.3 Deductive Systems

A deductive system for a temporal logic consists of a set of axiom schemes and inference rules. A formula p is said to be *provable*, written $\vdash$p, if there exists a finite sequence of formulae, ending with p such that each formula is an instance of an axiom or follows from previous formulae by application of one of the inference rules. A deductive system is said to be *sound* if every provable formula is valid. It is said to be *complete* if every valid formula is provable.

Consider the following axioms and rules of inference:

Axiom Schemes:

Ax1.	All validities of propositional logic
Ax2	$EFp \equiv E[\mathit{true}\ U\ p]$
Ax2b.	$AGp \equiv \neg EF\neg p$
Ax3.	$AFp \equiv A[\mathit{true}\ U\ p]$
Ax3b.	$EGp \equiv \neg AF\neg p$
Ax4.	$EX(p \vee q) \equiv EXp \vee EXq$
Ax5.	$AXp \equiv \neg EX\ \neg p$
Ax6.	$E(p\ U\ q) \equiv q \vee (p \wedge EXE(p\ U\ q))$
Ax7.	$A(p\ U\ q) \equiv q \vee (p \wedge AXA(p\ U\ q))$
Ax8.	$EX\mathit{true} \wedge AX\mathit{true}$
Ax9.	$AG(r \Rightarrow (\neg q \wedge EXr)) \Rightarrow (r \Rightarrow \neg A(p\ U\ q))$
Ax9b.	$AG(r \Rightarrow (\neg q \wedge EXr)) \Rightarrow (r \Rightarrow \neg AFq))$
Ax10.	$AG(r \Rightarrow (\neg q \wedge (p \Rightarrow AXr))) \Rightarrow (r \Rightarrow \neg E(p\ U\ q))$
Ax10b.	$AG(r \Rightarrow (\neg q \wedge AXr)) \Rightarrow (r \Rightarrow \neg EFq))$
Ax11.	$AG(p \Rightarrow q) \Rightarrow (EXp \Rightarrow EXq)$

Rules of Inference:

R1. if $\vdash p$ then $\vdash AGp$ (Generalization)

R2. if $\vdash p$ and $\vdash p \Rightarrow q$ then $\vdash q$ (Modus Ponens)

This deductive system for CTL is easily seen to be sound. We can also establish the following (cf. [EH85], [BPM81]):

Theorem 6.21. The above deductive system for CTL is complete.

Proof Sketch. Suppose p_0 is valid. Then $\sim p_0$ is unsatisfiable. We apply the above tableau-based decision procedure to $\sim p_0$. All nodes whose label includes $\sim p_0$ will be eliminated. In the sequel, we use the following notation and terminology. We use $\wedge s$ to denote the conjunction of all formulae labelling node s. We also write $p \in s$ for $p \in L(s)$, and we say that formula p is *consistent* provided that not $\vdash \sim p$.

Claim 1: If node s is deleted then $\vdash \sim(\wedge s)$.

Assuming the claim, we will show that $\vdash p_0$. We will use the formulae below, whose validity can be established by propositional reasoning:

$$\vdash q \equiv \vee\{\wedge s\text{: s is a node in the tableau and } q \in s\} \text{ for each formula } q \in ecl(p_0)$$
$$\equiv \vee\{\wedge s\text{: s is a node in the tableau and } q \in s \text{ and } \wedge s \text{ is consistent}\}$$

$$\vdash \mathit{true} \equiv \vee\{\wedge s\text{: s is a node in the tableau}\} \equiv \vee\{\wedge s\text{: s is a node in the tableau and } \wedge s \text{ is consistent}\}$$

Thus $\vdash\sim p_0 \equiv \vee\{\wedge s$: s is a node in the tableau and $not(p_0) \in s\}$. Because $\sim p_0$ is unsatisfiable the decision procedure will delete each node s containing p_0 in its label. By Claim 1 above, for each such node s that is eliminated, $\vdash not(\wedge s)$. Thus we get $\models\sim\sim p_0$ and also $\vdash p_0$.

Before proving Claim 1, we establish

Claim 2: If (s,t) $\notin$ R as originally constructed then $\wedge$s $\wedge$ EX$\wedge$t is inconsistent.

Proof: Suppose (s,t) $\notin$ R. Then for some formulae AXp, AXp $\in$ s and $\sim$p $\in$ t. Thus, we can prove the following

a.	$\vdash \wedge s \Rightarrow AXp$	(since AXp $\in$ s)
b.	$\vdash \wedge t \Rightarrow \sim p$	(since $\sim$p $\in$ t)
c.	$\vdash AG(\wedge t \Rightarrow \sim p)$	(generalization rule)
d.	$\vdash EX\wedge t \Rightarrow EX\neg p$	(Ax11: monotonicity of EX operator)
e.	$\vdash (\wedge s \wedge EX\wedge t) \Rightarrow AXp \wedge EX\sim p$	(lines a,d and propositional reasoning)
f.	$\vdash (\wedge s \wedge EX\wedge t) \Rightarrow$ *false*	(Ax5 and def. AX operator)
g.	$\vdash \sim(\wedge s \wedge EX\wedge t)$	(propositional reasoning)

Thus we have established that $\wedge$s $\wedge$ EX$\wedge$t is inconsistent, thereby completing the proof of claim 2.

We now are ready to give the proof of Claim 1. We argue by induction on when a node is deleted that, if node s is deleted then $\vdash \sim\wedge$s.

Case 1: If $\wedge$s is consistent, then s is not deleted on account of having no successors.

To see this, we note that we can prove

$$\begin{aligned}\vdash \wedge s \quad &\equiv \wedge s \wedge EX\mathit{true}\\ &\equiv \wedge s \wedge EX(\vee\{\wedge t\text{: } \wedge t \text{ is consistent}\})\\ &\equiv \wedge s\ (\vee\{EX\wedge t\text{: } \wedge t \text{ is consistent}\})\\ &\equiv \vee\{\wedge s \wedge EX\wedge t\text{: } \wedge t \text{ is consistent}\}\end{aligned}$$

Thus if $\wedge$s is consistent, $\wedge$s $\wedge$ EX$\wedge$t is consistent for some t. By Claim 1 above (s,t) $\in$ R in the original tableau. By induction, hypothesis, node t is not eliminated. Thus, (s,t) $\in$ R in the current tableau and node s is not eliminated due to having no successors.

Case 2: Node s is eliminated on account of EXq $\in$ s, but s has no successor t with q $\in$ t.

This is established using an argument like that in case 1.

Case 3: Node s is deleted on account of EFq $\in$ s, which is not fulfilled (ranked) at s.

Let V = {t : EFq $\in$ t but is not fulfilled} $\cup$ {t : EFq $\notin$ t}. Note that node s $\in$ V. Moreover, the complement of V = {t: EFq $\in$ t and is fulfilled}.

Let r = $\vee$\{$\wedge$ t: t $\in$ V\}. We claim that $\vdash r \Rightarrow (\neg q \wedge AXr)$. It is clear that $\vdash r \Rightarrow \neg q$, because $\neg q \in t$ for each t $\in$ V and $\vdash \wedge t \Rightarrow \neg q$. We must now show that $\vdash r \Rightarrow AXr$. It suffices to show that for each t $\in$ V, $\vdash \wedge t \Rightarrow AXr$. Suppose not. Then $\exists$ t $\in$ V, $\wedge$t $\wedge$ EX$\sim$r is consistent. Since $\neg r = \vee\{\wedge t'\text{: } t' \notin V\}$, $\exists$ t $\in$ V $\exists$ t$'$ $\notin$ V, $\wedge$t $\wedge$ EX$\wedge$t$'$ is consistent. By claim 2 above, (t,t$'$) $\in$ R as originally constructed, and since $\wedge$t and $\wedge$t$'$ are each consistent neither is eliminated, by induction hypothesis. So (t,t$'$) $\in$ R in the current tableau. Since t$'$ $\notin$ V, EFq $\in$ $'$ and is ranked. But by virtue of the arc (t,t$'$) in the tableau, t should also be ranked for EFq, a contradiction to t being a member of V. Thus $\vdash r \Rightarrow AXr$.

By generalization $\vdash$ AG(r $\Rightarrow$ AXr) and by the induction axiom for EF and modus ponens, $\vdash$ r $\Rightarrow$ ¬EFq. Now $\vdash$ ∧s $\Rightarrow$ r, by definition of r (as the disjunction of formulae for each state in V, which includes node s). However, we had assumed EFq ∈ s which of course means that $\vdash$ ∧s $\Rightarrow$ EFq. Thus $\vdash$ ∧s $\Rightarrow$ *false*, so that ∧ s is inconsistent.

The proofs for the other cases for eventualities E(p U q), AFq, and A(p U q) are similar to that for case 3. □

6.4 Model Checking

The model checking problem (roughly) is: Given a finite structure M and a propositional TL formula p, does M define a model of p? For most any propositional TL the model checking problem is decidable since we can do, if needed, an exhaustive search through the paths of the finite input structure. The problem has important applications to mechanical verification of finite state concurrent systems (see section 7.3). The significant issues from the theoretical standpoint are to analyze and classify logics with respect to the complexity of model checking. For some logics, which have adequate expressive power to capture certain important correctness properties, we can develop very efficient algorithms for model checking. Other logics cannot be model checked so efficiently.

We say roughly because there is some potential ambiguity in the above definition. What system of TL is the formula p from? In particular, it is branching or linear time? Also, what does it mean for a structure M to be a model of a formula p? From the definition of satisfiability for a formula p_0 of branching time logic, a state formula, it seems that we should say a structure M is a model of a formula p_0 provided it contains a state s such that M,s $\models$ p_0. From the technical definition of satisfiability for a formula p_0 of linear time logic, it appears we should say a structure M is a model of a formula p_0 provided it contains a fullpath x such that M,x $\models$ p_0. However, the number of fullpaths can be exponential in the size of a finite structure M. It thus seems that the complexity of model checking for linear time could be very high, since in effect an examination of all paths through the structure could be required.

To overcome these difficulties, we therefore formalize the model checking problem as follows:

The *Branching Time Logic Model Checking Problem (BMCP)* formulated for propositional branching time logic BTL is: Given a finite structure M=(S,R,L) and a BTL formula p, determine for each state s in S whether M,s $\models$ p and, if so, label s with p. The *Linear Time Logic Model Checking Problem (LMCP)* for propositional linear time logic LTL can be similarly formulated as follows: Given a finite structure M=(S, R, L) and an LTL formula p, determine for each state in S, whether there is a fullpath satisfying p starting at s, and, if so, label s with Ep.

This definition of LMCP may, at first glance, appear to be incorrectly formulated because it defines truth of linear time formulae in terms of states. However, one should note that there is a fullpath in finite structure M satisfying linear time formula p_0 iff there is such a fullpath starting at some state s of M. It thus suffices to solve LMCP and then scan the states to see if one is labelled with Ep. We can also handle the applications-oriented convention that linear time formula p is true of a structure (representing a concurrent program) iff it is true of all (initial) paths in the structure, because p is true of all paths in the structure iff Ap holds at all states of the structure.

Since $Ap \equiv \neg E \neg p$, by solving LMCP and then scanning all (initial) states to check whether Ap holds, we get a solution to the applications formulation.

We now analyze the complexity of model checking linear time. The next three results are from [SC85]:

Lemma 6.22. The model checking problem for PLTL is polynomial time reducible (transformable) to the satisfiability problem for PLTL.

Proof Sketch. The key idea is that we can readily encode the organization of a given finite structure into a PLTL formula. Suppose $M = (S,R,L)$ is a finite structure and p_0 a PLTL formula, over underlying set of atomic propositions AP. Let AP$'$ be an extension of AP obtained by including a new, "fresh" atomic proposition Q_s for each state $s \in S$. The local organization of M at each state s is captured by the formula

$$q_s = Q_s \Rightarrow (\wedge_{P \in L(s)} P \wedge \wedge_{P \notin L(s)} \wedge \vee_{(s,t) \in R} X Q_t$$

while the formula below asserts that the above local organization prevails globally:

$$q' = G((\Sigma_{s \in S} Q_s = 1) \wedge \wedge_{s \in S} q_1)$$

and means, in more detail, that exactly one Q_s is true at each time and that the corresponding q_s holds.

Claim: There exists a fullpath x_1 in M such that $M,x_1 \models p_0$ iff $q' \wedge p_0$ is satisfiable.

The $\rightarrow$ direction is clear: annotate M with propositions from AP$'$. The path x_1 so annotated is model of $q' \wedge p_0$.

The $\leftarrow$ direction can be seen as follows. Suppose $M',x \models q' \wedge p_0$. The $x = u_0,u_1,u_2,\ldots$ matches the organization of M in that, for each i (a) with state u_i we associate a state s of M—the unique one such that $M',u_i \models P_s$—that satisfies the same atomic propositions in AP as does s; call it $s(u_i)$ and (b) the successor u_{i+1} along x of u_i is associated with a state $t = s(u_{i+1})$ of M which is a successor of s in M. Thus, the path $x_1 = s(u_0),s(u_1),s(u_2),\ldots$ in M is such that $M,x_1 \models p_0$. □

Theorem 6.23. The model checking problem for PLTL is PSPACE-complete.

Proof Idea. Membership in PSPACE follows from the preceding lemma and the theorem establishing that satisfiability is in PSPACE. PSPACE-hardness follows by a generic reduction from PSPACE Turing machines. □

Remark: The above PSPACE-completeness result holds for PLTL(F,X), the sublanguage of PLTL obtained by restricting the temporal operators to just X, F, and its dual G. It also holds for PLTL(U), the sublanguage of PLTL obtained by restricting the temporal operators to just U and its dual B.

Theorem 6.24. The problem of model checking for PLTL(F) is NP-complete.

Proof Idea. To establish membership in NP, we design a nondeterministic algorithm that guesses a finite path in the input structure M leading to a strongly connected component, such that any unwinding of the component prefixed by some finite path comprises a candidate model of the input formula p_0. To check that it is indeed a model evaluate each subformula of each state of the candidate model, which can be done in polynomial time. NP-hardness follows by a reduction from 3-SAT. □

We now turn to model checking for branching time logic. First we have from [CE81]:

Theorem 6.25. The model checking problem for CTL is in deterministic polynomial time.

This result is somewhat surprising since CTL seems somehow more complicated than the linear time logic PLTL. Because of such seemingly unexpected complexity results, the question of the complexity of model checking has been an issue in the branching versus linear time debate. Branching time, as represented by CTL, appears to be more efficient than linear time, but at the cost of potentially valuable expressive power, associated with, for example, fairness.

However, the real issue for model checking is not branching versus linear time, but simply what are the basic modalities of the branching time logic to be used. Recall that the basic modalities of a branching time logic are those of the form Ap or Ep, where p is a "pure" linear time formula containing no path quantifiers itself. Then we have the following result of [EL85]:

Theorem 6.26. Given any model checking algorithm for a linear logic LTL there is a model checking algorithm for the corresponding branching logic BTL, whose basic modalities are defined by the LTL, of the same order of complexity.

Proof idea. Simply evaluate nested branching time formulae Ep or Ap by recursive descent. For example, to model check EFAGP, recursively model check AGP, then label every state labelled with AGP with fresh proposition Q and model check EFQ. □

For example, CTL* can be reduced to PLTL since the basic modalities of CTL* are of the form A or E followed by a PLTL formula. As a consequence we get (cf. [CES83]):

Corollary 6.27. The model checking problem for CTL* is PSPACE-complete.

Thus the increased expressive power of the basic modalities of CTL* incurs a significant complexity penalty. However, it can be shown that basic modalities for reasoning under fairness assumptions do not cause complexity difficulties for model checking. These matters are discussed further in Section 7.

6.5 Automata on Infinite Objects

There has been a resurgence of interest in finite state automata on infinite objects, due to their close connection to TL. They provide an important alternative approach to developing decision procedures for testing satisfiability for propositional temporal logics. For linear time temporal logics the tableau for formula p_0 can be viewed as defining a finite automaton on infinite strings that essentially accepts a string iff it defines a model of the formula p_0. The satisfiability problem for linear logics is thus reduced to the emptiness problem of finite automata on infinite strings. In a related but somewhat more involved fashion, the satisfiability problem for branching time logics can be reduced to the nonemptiness problem for finite automata on infinite trees.

For some logics, the only known decision procedures of elementary time complexity (i.e., of time complexity bounded by the composition of a fixed number of exponential functions), are obtained by reductions to finite automata on infinite trees. The use of automata transfers some difficult combinatorics onto the automata-theoretic machinery. Investigations into such automata-theoretic decision procedures is an active area of research interest.

We first outline the automata-theoretic approach for linear time. As suggested by Theorem 6.16, the tableau construction for CTL can be specialized, essentially by dropping the path quantifiers to define a tableau construction for PLTL. The extended closure of a PLTL formula p_0, ecl(p_0), is defined as for CTL, remembering that in a linear structure, $Ep \equiv Ap \equiv p$. The notions of maximal, and propositionally consistent subsets of of ecl(p_0) are also defined analogously. The (initial) tableau for p_0 is then a structure T = (S,R,L) where S is the set of maximal, propositionally consistent subsets of ecl(p_0), i.e., states, $R \subseteq S \cdot S$ consists of the transitions (s,t) defined by the rule $(s,t) \in R$ exactly when $\forall$ formula $Xp \in ecl(p_0)$, $Xp \in s$ iff $p \in t$, and L(s) = s, for each $s \in S$.

We may view the tableau for PLTL formula p_0 as defining the transition diagram of a nondeterministic finite state automaton $\mathcal{A}$ which accepts the set of infinite strings over alphabet Σ = PowerSet(AP) that are models of p_0, by letting the arc (u,v) be labelled with AtomicPropositions(v), i.e., the set of atomic propositions in v. Technically, $\mathcal{A}$ is a tuple of the form $(S \cup \{s_0\},\Sigma, \delta, s_0, —)$ where $s_0 \notin S$ is a unique start state, δ is defined so that $\delta(s_0,a) = \{$ states $s \in S$: $p_0 \in s$ and AtomicPropositions(s) = a$\}$ for each $a \in \Sigma$, $\delta(s,a) = \{$ states $t \in S$: $(s,t) \in R$ and AtomicPropositions(s) = a$\}$. The acceptance condition is defined below. A *run* r of $\mathcal{A}$ on input $x = a_1a_2a_3... \in \Sigma^\omega$ is an infinite sequence of states $s_0s_1s_2...$ such that $\forall i \geq 0$ $\delta(s_i,a_{i+1}) \supseteq \{s_{i+1}\}$. Note that $\forall i \geq 1$ AtomicPropositions(s_i) = a_i. Any run of $\mathcal{A}$ would correspond to a model of p_0, in that $\forall i \geq 1$, $x^i \models \wedge\{$ formulae p: $p \in s_i\}$, except that eventualities might not be fulfilled. To check fulfillment, we can easily define acceptance in terms of complemented pairs (cf. [Th89]). If ecl(p_0) has m eventualities $(p_1 U q_1),...,(p_m U q_m)$, we let $\mathcal{A}$ have m pairs (RED$_i$, GREEN$_i$) of lights. Each time a state containing $(p_i U q_i)$ is entered, flash RED$_i$; each time a state containing q_i is entered flash GREEN$_i$. A run r is accepted iff for each $i \in [1:m]$, there are infinitely many RED$_i$ flashes implies there are infinitely many GREEN$_i$ flashes iff every eventuality is fulfilled iff the input string x is a model of p_0.

We can convert $\mathcal{A}$ into an equivalent nondeterministic Buchi automaton $\mathcal{A}_1$, where acceptance is defined simply in terms of a single GREEN light flashing infinitely often. We need some terminology. We say that the eventuality (p U q) is *pending* at state s of run r provided that $(p U q) \in s$ and $q \notin s$. Observe that run r of $\mathcal{A}$ on input x corresponds to a model of p_0 iff not($\exists$ eventuality $(p U q) \in ecl(p_0)$, (p U q) is pending almost everywhere along r) iff $\forall$ eventuality $(p U q) \in ecl(p_0)$, (p U q) is not pending infinitely often along r. The Buchi automaton $\mathcal{A}_1$ is then obtained from $\mathcal{A}$ augmenting the state with an m+1 valued counter. The counter is incremented from i to i+1 mod (m + 1) when the ith eventuality, $(p_i U q_i)$ is next seen to be not pending along the run r. When the counter is reset to 0, flash GREEN and set the counter to 1. (If m = 0, flash GREEN is every state.) Now observe that there are infinitely many GREEN flashes iff $\forall i \in [1:m]$ $(p_i U q_i)$ is not pending infinitely often iff every pending eventuality is eventuality fulfilled iff the input string x defines a model of p_0. Moreover, $\mathcal{A}_1$ still has $\exp(|p_0|) \cdot O(|p_0|) = \exp(|p_0|)$ states.

Similarly, the tableau construction for a branching time logic with relatively simple modalities such as CTL can be viewed as defining a Buchi tree automaton that, in essence, accepts all models of a candidate formula p_0. (More precisely, every tree accepted by the automaton is a model of p_0, and if p_0 is satisfiable there is some tree accepted by the automaton.) General automata-theoretic techniques for reasoning about a number of relatively simple logics, including CTL, using Buchi tree automata have been described by Vardi and Wolper [VW84].

For branching time logics with richer modalities such as CTL*, the tableau construction is not directly applicable. Instead, the problem reduces to constructing a tree automaton for the branching time modalities (such as Ap) in terms of the string automaton for the corresponding linear time formula (such as p). This tree automaton will in general involve a more complicated acceptance condition such as pairs or complemented pairs, rather than the simple Buchi condition. Somewhat surprisingly, the only known way to build the tree automaton involves difficult combinatorial arguments and/or appeals to powerful automata-theoretic results such as McNaughton's construction ([McN66]) for determinizing automata on infinite strings.

The principal difficulty manifests itself with just the simple modality Ap. The naive approach of building the string automaton for p and then running it down all paths to get a tree automaton for Ap will not work. The string automaton for p must be determinized first. To see this, consider two paths xy and xz in the tree which start off with the same common prefix x but eventually separate to follow two different infinite suffixes y or z. It is possible that p holds along both paths, but in order for the nondeterministic automaton to accept, it might have to "guess" while reading a particular symbol of x whether it will eventually read the suffix y or the suffix z. The state it guesses for y is in general different from the state it guesses for z. Consequently, no single run of a tree automaton based on a nondeterministic string automaton can lead to acceptance along all paths.

For a CTL* formula of length n, use of classical automata-theoretic results yields an automaton of size triple exponential in n. (Note: by triple exponential we mean exp(exp(exp(n))), etc.) The large size reflects the exponential cost to build the string automaton as described above for a linear time formula p plus the double exponential cost of McNaughton's construction to determinize it. Nonemptiness of the automaton can be tested in exponential time to give a decision procedure of deterministic time complexity quadruple exponential. in n. In [ESi84] it was shown that, due to the special structure of the string automata derived from linear temporal logic formulae, such string automata could be determinzed with only single exponential blowup. This reduced the complexity of the CTL* decision procedure to triple exponential. Further improvement is possible as described below.

The size of a tree automaton is measured in terms of two parameters: the number of states and the number of pairs in the acceptance condition. A careful analysis of the tree automaton constructions in temporal decision procedures shows that the number of pairs is logarithmic in the number of states, and for CTL* we get an automaton with double exponential states and single exponetial pairs. An algorithm of [EJ88] shows how to test nonemptiness in time polynomial in the number of states, while exponential in the number of pairs. For CTL* this yields a decision procedure of deterministic double exponential time complexity, matching the lower bound of [VS85].

One drawback to the use of automata is that, due to the delicate combinatorial constructions involved, there is usually no clear relationship between the structure of the automaton and the syntax of the candidate formula. An additional drawback is that in such cases, the automata-theoretic approach provides no aid in finding sound and complete axiomitizations. For example, the existence of an explicit, sound and complete axiomitization for CTL* has been an open question for some time. (Note: We refer here to an axiomitization for its validities over the usual semantics generated by a binary relation; interestingly, for certain nonstandard semantics, complete axiomitizations are known (cf. [Ab80], [LS84]).)

However, there are certain definite advantages to the automata-theoretic approach. First, it does provide the only known elementary time decision procedures for some logics. Secondly, automata can provide a general, uniform framework encompassing temporal reasoning (cf. [VW5] [VW86] [V87]). Automata themselves have been proposed as a potentially useful specification language. Automata, moreover, bear an obvious relation to temporal structures, abstract concurrent programs, etc. This makes it possible to account for various types of temporal reasoning applications such as program synthesis and mechanical verification of finite state programs in a conceptually uniform fashion. Verification systems based on automata have also been developed (cf. [Ku86]).

We note that not only has the field of TL benefited from automata theory, but the converse holds as well. For example, the tableau concept for the branching time logic CTL, particularly the state/prestate formulation, suggests a very helpful notion of the transition diagram for a tree automaton (cf. [Em85]). This has made it possible to apply tableau-theoretic techniques to automata, resulting in more efficient algorithms for testing nonemptiness of automata, which in turn can be used to get more efficient decision procedures for satisfiability of TL's (cf. [EJ88]). Still another improved nonemptiness algorithm, motivated by program synthesis applications is given in [PR89]. New types of automata on infinite objects have also been proposed to facilitate reasoning in TL's (cf. [St81], [VS85], [MP87a]). A particularly important advance in automata theory motivated by TL is Safra's construction ([Sa88]) for determinizing an automaton on infinite strings with only a single exponential blowup, without regard to any special structure possessed by the automaton. Not only is Safra's construction an exponential improvement over McNaughton's construction but it is conceptually much more simple and elegant. In this way we see that not only can TL sometimes benefit from adopting the automata-theoretic viewpoint, but also conversely and even synergistically, the study of automata on infinite objects has been advanced by work motivated by and using the techniques of TL.

7 The Application of Temporal Logic to Program Reasoning

Temporal Logic has been suggested as a formalism especially appropriate to reasoning about ongoing concurrent programs, such as operating systems, which have a *reactive* nature, as explained below (cf. [Pn86]).

We can identify two different classes of programs (also referred to as systems). One class consists of those ordinarily described as "sequential" programs. Examples include a program to sort a list, programs to implement a graph algorithm as discussed in, say, the chapter on graph algorithms, and programs to perform a scientific calculation. What these programs have in common is that they normally terminate. Moreover, their behavior has the following pattern: they initially accept some input, perform some computation, and then terminate yielding final output. For all such systems, correctness can be expressed in terms of a Precondition/Postcondition pair in a formalism such as Hoare's logic or Dijkstra's weakest preconditions, because the systems' underlying semantics can be viewed as a *transformation* from initial states to final states, or from Postconditions to Preconditions.

The other class of programs consists of those which are continuously operating, or, ideally, nonterminating. Examples include operating systems, network communication protocols, and air traffic control systems. For a continuously operating program its normal behavior is an arbitrarily long, possibly nonterminating computation, which maintains an ongoing interaction with the environment. Such programs can be described as *reactive* systems. The key point concerning such systems is that they maintain an ongoing interaction with the environment, where intermediate outputs of the program can influence subsequent intermediate inputs to the program. Reactive systems thus subsume many programs labelled as concurrent, parallel, or distributed, as well as process control programs. Since there is in general no final state, formalisms such as Hoare's logic which are based on an initial state-final state semantics, are of little use for such reactive programs. The operators of temporal logic such as *sometimes* and *always* appear quite appropriate for for describing the time-varying behavior of such programs.

What is the relationship between concurrency and reactivity? They are in some sense independent. There are transformational programs that are implemented to exploit parallel architectures (usually, to speed processing up, allowing the output to be obtained more quickly). A reactive system could also be implemented on a sequential architecture.

On the other hand, it can be recommended that in general concurrent programs should be viewed as reactive systems. In a concurrent program consisting of two or more processes running in parallel, each process is generally maintaining an ongoing interaction with its environment, which usually includes one or more of the other processes. If we take the compositional viewpoint, where the meaning of the whole is defined in terms of the meaning of its parts, then the entire system should be viewed in the same fashion as its components, and the view of any system is a reactive one. Even if we are not working in a compositional framework, the reactive view of the system as a whole seems a most natural one in light of the ongoing behavior of its components. Thus, in the sequel when we refer to a concurrent program, we mean a reactive, concurrent system.

There are two main schools of thought regarding the application of TL to reasoning about concurrent programs. The first might be characterized as "proof-theoretic." The basic idea is to manually compose a program and a proof of its correctness using a formal deductive system, consisting of axioms and inference rules, for an appropriate temporal specification language. The second might be characterized as "model-theoretic." The idea here is to use decision procedures that manipulate the underlying temporal models corresponding to programs and specifications to automate the tasks of program construction and verification. We subsequently outline the approach of each of these two schools. First, however, we discuss the types of correctness properties of practical interest for concurrent programs and their specification in TL.

7.1 Correctness Properties of Concurrent Programs

There are a large number of correctness properties that we might wish to specify for a concurrent program. These correctness properties usually fall into two broad classes (cf. [Pn77], [OL82]). One class is that of "safety" properties also known as "invariance" properties. Intuitively, a safety property asserts that "nothing bad happens." The other class consists of the "liveness" properties also referred to as "eventuality" properties or "progress" properties. Roughly speaking, a liveness property asserts that "something good will happen." These intuitive descriptions of safety and liveness are made more precise below, following [Pn86].

A *safety* property states that each finite prefix of a (possibly infinite) computation meets some requirement. Safety properties are thus those that are (initially) equivalent to a formula of the form Gp, for some past formula p. The past formula describes the condition required of finite prefixes, while the G operator ensures that p holds of all finite prefixes. Note that this formal definition of safety requires that always "nothing bad has happened yet," consistent with the intuitive characterization of [OL82] mentioned above.

Any formula built-up from past formulae, the propositional connectives $\wedge$ and $\vee$, and the future temporal operators G and U_w can be shown to express a safety property. For example, $(p\ U_w\ q) \equiv_i G(G^- p \vee F^-(q \wedge X^- G^- p)$.

A number of concrete examples of safety properties can be given. The partial correctness of a program with respect to a precondition ϕ and postcondition ψ, which stipulates that if program execution begins in a state satisfying ϕ, then if it terminates the final state satisfies ψ, is expressed by

$$atl_0 \wedge \phi \Rightarrow G(atl_h \Rightarrow \psi)$$

where the program's start label is l_0 and its halt label is l_h. (Note: this formula is initially equivalent to $G(F^-(\neg(atl_0 \wedge \phi) \wedge X_w^- \textit{false})) \vee G(atl_h \Rightarrow \psi))$ thereby demonstrating that it is safety property according to the technical definition.)

Other safety properties include *global invariance* of assertion p is expressed simply by Gp. To capture *local invariance* which means that p holds whenever control is at location l, we write G(atl $\Rightarrow$ p).

The requirement of *mutual exclusion* for a two process solution to the critical section problem can be written

$$G(\ \neg(atCS_1 \wedge atCS_2))$$

where $atCS_i$ indicates that control of process i is at its critical section.

Another very important property for concurrent programs is *freedom from deadlock*. A concurrent program is *deadlocked* if no process is enabled to proceed. The formula $G(\ \textit{enabled}_1 \vee \ldots \vee \textit{enabled}_m\ )$ captures freedom from deadlock for a concurrent program with m processes.

Liveness properties are in some sense dual to safety properties, requiring that some finite prefix property hold a certain number of times.

The *basic liveness* properties are technically defined to be those (initially) expressible in the form Fp, $\overset{\infty}{F}$p, or $\overset{\infty}{G}$p, where p is a past formula required to hold for some, for infinitely many, or for all but but a finite number, resp., of the finite prefixes of a computation. It is interesting to note that $(p\ U_s\ q) \equiv_i F(q \wedge X_w^-\ G^- p)$ for any past formulae p and q, thus showing the strong until to be a basic liveness property, even though it is not immediately obvious that it can be expressed in the required form. Also note that $Fp \equiv_i GF(F^- p) \equiv_i FG(F^- p)$ and is technically redundant, even though we find it more convenient to keep Fp separated out. A more serious redundancy is that, by our definition, each safety property is a basic liveness property, since $Gp \equiv_i GF(G^- p)$ for any past formula p.

If we wish to avoid this redundancy, we can first define an *invincible* past formulae to be one such that every finite sequence x has a finite extension x' with $(x', \text{length}(x')) \models p$ (i.e., with p holding at the last state of x').

We then define the *pure liveness* properties to be those initially equivalent to one of the formulae Fp, GFp, FGp, for some invincible past formula p. Note that any satisfiable state formula p is an invincible past formula, so that the pure liveness formulae still include a broad range of properties. However, $(p\ U_s\ q)$ is not a pure liveness property, because while $(p\ U_s\ q) \equiv_i F(q \wedge X^- F^- p)$, the formula $q \wedge X^- F^- p$ is not invincible. It is expressible as the conjunction of a safety property and a pure liveness property: $(p\ U_s\ q) \equiv_i (p\ U_w\ q) \wedge Fq$.

Note that if p is a pure liveness property, then it has the following characteristic: every finite sequence x can be extended to a finite or infinite sequence x' such that $(x', 0) \models p$. This corresponds to the intuitive characterization of liveness, that "something good will happen," of [OL82].

Further work on syntactic and semantic characterizations of safety and liveness properties are given in [AS85] and [Si85.]

One important generic liveness property has the form

$$G(p \Rightarrow Fq)$$

for past formulae p and q, and is called *temporal implication* (cf. [Pn77], [La80]). Many specific correctness properties are instances of temporal implication, as described below.

An *intermittent assertion* is expressed by

$$G(\ (atl \wedge \phi) \Rightarrow F(atl' \wedge \phi'))$$

meaning that whenever ϕ is true at location l, then ϕ' will eventually be true at location l' (cf. [Bu74], [MW78]). An important special type of intermittent assertion is *total correctness* of a program with respect to a precondition ϕ and postcondition ψ. It is expressed by

$$atl_0 \wedge \phi \Rightarrow F(atl_h \wedge \psi)$$

which indicates that if the program starts in a state satisfying ϕ, then it halts in a state satisfying ψ.

The property of *guaranteed accessibility* for a process in a solution to the mutual exclusion problem to enter its critical section, once it has indicated that it wishes to do so is expressed by

$$G(atTry_i \Rightarrow FatCS_i)$$

where $atTry_i$ and $atCS_i$ indicated that process i is in its Trying section or Critical section, respectively. This property is sometimes referred to as *absence of individual starvation for process i.* General guaranteed accessibility is of the form

$$G(atl \Rightarrow Fatl')$$

Still another property expressible in this way is *responsiveness.* Consider a system consisting of a resource controller that monitors access to a shared resource by competing user processes. We would like to ensure that each request for access eventually leads to a response in the form of a granting of access. This is captured by an assertion of the form $G(req_i \Rightarrow Fgrant_i)$ where req_i and $grant_i$ are predicates indicating that a request by process is made or a grant of access to process i is given, respectively.

The fairness properties discussed in Section 5 are also liveness properties.

A final general type of correctness property is informally known as the *precedence* properties. These properties have to do do with temporal ordering, precedence, or priority of events. We shall not give a formal definition but instead illustrate the class by several examples.

To express *absence of unsolicited response* as in the resource controller example above, where we want a $grant_i$ to be issued only if preceded by a req_i we can write

$$\neg grant_i \Rightarrow (\neg grant_i \ U_w \ req_i).$$

Alternatively, we can write $(req_i \ B \ grant\ _i)$, where we recall that the precedes operator $(p \ B \ q)$ asserts that the first occurrence of q, if any, is strictly preceded by an occurrence of p.

The important property of First-In-First-Out(FIFO) responsiveness can be written in a straightforward but slightly imprecise fashion as

$$(req_i \ B \ req_j) \Rightarrow (grant_i \ B \ grant_j).$$

A more accurate expression is

$$(req_i \land \neg req_j \land \neg grant_j) \Rightarrow ((\neg grant_j) \ U_w \ grant_i)$$

where we rely on the assumption that once a request has been made, it is not withdrawn before it has been granted. Hence, $req_i \land \neg req_j$ implies that process i's request preceded that of process j.

It is interesting to note the importance of correctly formalizing in the formal specification language our intuitive understanding of the problem. An important application where this issue arises is the specification of correct behavior for a message buffer. Such buffers are often used in distributed systems based on message passing, where one process transmits messages to another process via an intermediate, asynchronous buffer that temporarily stores messages in transit.

We assume that the buffer has an input channel x and output channel y. It also has unbounded storage capacity and is assumed to operate according to FIFO discipline. We want to specify that the log of input/output transactions for the buffer is correct, viz., that the sequence of messages output on channel y equals to the sequence of messages input on channel x.

An important limitation of PLTL and related formalisms was established by Sistla et. al. [SCFM84] which shows that an unbounded FIFO buffer cannot be specified in PLTL. Essentially, the problem is that any particular formula p of PLTL is of a fixed size and corresponds to a bounded size finite state automaton, while the buffer can hold an arbitrarily large sequence of messages, thereby permitting the finite automaton to become "confused." Moreover, the problem is not alleviated by extending the formalism to be pure (i.e., uninterpreted) FOLTL (cf. [Ko87]).

However, as noted in [SCFM84] there exist partially interpreted FOLTL's which make it possible to capture correct behavior for a message buffer. One such logic provides history variables that accumulate the string of all previous states along with a prefix predicate ($\leq$) on these histories. The safety portion of the specification is given by $G(y \leq x)$ which asserts that the sequence of messages output is always a prefix of the sequence of messages input. The liveness requirement is expressed by $\forall z\ G(x=z \Rightarrow F(y=z))$ which ensures that whatever sequence appears along the input channel is eventually replicated along the output channel.

The essential feature of the above specification based on histories is the ability to, in effect, associate a unique sequence number with each message, thereby ensuring that all messages are distinct. Using in(m) to indicate that message m is placed on input channel x and out(m) for the placement of message on output channel y, we have the following alternative specification in the style of [Ko87]): The formula

$$\forall m\ G(in(m)\ B\ out(m))$$

specifies that any message output must have been previously input.

The formula

$$\forall m \forall m'\ G(in(m) \wedge XFin(m') \Rightarrow F(\ out(m) \wedge XFout(m')))$$

asserts that FIFO disciple is maintained, i.e. messages are output in the same order they were input.

The liveness requirement is expressed by

$$\forall m\ G(in(m) \Rightarrow Fout(m))$$

while the assumption of message uniqueness is captured by

$$\forall m\ \forall m'\ G(\ (in(m) \wedge XFin(m')) \Rightarrow (m \neq m')\ )$$

Note that the requirement of message uniqueness is essential for the correctness of the specification. Without it, a computation with, e.g., the same message output twice for each input message would be permitted.

Recently, Wolper [Wo86] has provided additional insight into the power of logical formalisms for specifying message buffers. First, he pointed out that PLTL is *a priori* inadequate for specifying message buffers when the underlying data domain is infinite, since each PLTL formula is finite. However, he goes on to show that PLTL is nonetheless adequate for specifying message buffer protocols that the *data independence* criterion, which requires that the behavior of the protocol does not depend on the value or content of a message. While it is in general undecidable whether a protocol is data independent, a simple syntactic check of the protocol, if positive, ensures data independence. This amounts to checking that the only possible operation performed on message contents are reading from channels to variables, writing from variables to channels, and copying between variables.

It is shown in [Wo86] that it is enough for data independent buffer protocols to assert correctness over a 3 symbol message alphabet $\Sigma = \{m_1, m_2, m_3\}$, so that the input is of the form $m_3{}^* m_1 m_3{}^* m_2 m_3{}^\omega$ iff the output is of the form $m_3{}^* m_1 m_3{}^* m_2 m_3{}^\omega$. This matching of output to input can be expressed in PLTL, using propositions in_m_i and out_m_i (assumed to be exclusive and exhaustive), $1 \leq i \leq 3$, to indicate the appearance of message m_i on the input channel and on the output channel, respectively, as

$$\begin{aligned}&(\ (in\_m_3 \ U \ (in\_m_1 \wedge X(in\_m_3 \ U \ (in\_m_2 \wedge XGin\_m_3)))) \Rightarrow \\ &\quad (out\_m_3 \ U \ (out\_m_1 \wedge X(out\_m_3 \ U \ (out\_m_2 \wedge XGout\_m_3)))) \ ) \\ &\wedge \wedge_{i=1\ldots3} \ (out\_m_i \ B \ in\_m_i).\end{aligned}$$

Intuitively this works because it ensures that each pair of distinct input messages are transmitted through to the output correctly; since the buffer is assumed to be oblivious to the message contents, the only way it can ensure such correct transmission for the three symbol alphabet is to transmit correctly over any alphabet, including those with distinct messages.

The reader may have noticed that the above example specifications were given in linear TL. If we wished to express them in branching TL we would merely need to prefix each assertion by the universal path quantifier. The reason linear TL sufficed was that above we were mainly interested in properties holding of ıall computations of a concurrent program. If we want to express lower bounds on nondeterminism and/or concurrency we need the ability to use existential path quantification, provided only by branching time logic. Such lower bounds are helpful in applications such as program synthesis. Moreover, branching time makes it possible to distinguish between *inevitability* of predicate P, which is captured by AFP, and *potentiality* of predicate P, which is captured by EFP. It also ensures that our specification logic is closed under semantic negation so that we can express, for example, not only absence of deadlock along all futures but also the possibility of deadlock along some future (cf. [La80], [EH86], [Pn85]).

7.2 Verification of Concurrent Programs: Proof-Theoretic Approach

A great deal of work has been done investigating the proof-theoretic approach to verification of concurrent programs using TL (cf. e.g. [Pn81], [MP81], [MP82], [MP83], [La 80], [Ha81], [OL82], [La83], [SMS82]). Typically, one tries to prove, by hand, that a given program meets a certain TL specification using various axioms and inference rules for the system of TL. A drawback of this approach is that proof construction is often a difficult and tedious task, with many details that require considerable effort and ingenuity to organize in an intellectually manageable fashion. The advantage is that human intuition can provide useful guidance that would be unavailable in a (purely) mechanical verification system. It should also be noted that the emphasis of this work has been to develop axioms, rules, and techniques that are useful in practice, as demonstrated on example programs, as opposed to meta-theoretic justifications of proof systems.

A proof system in the LTL framework has been given by Manna and Pnueli [MP83] consisting of three parts (i) A *general* part for reasoning about temporal formulae valid over all interpretations. This includes PLTL and FOLTL; (ii) A *domain* part for reasoning about variables and data structures over specific domains, such as the natural numbers, trees, lists, etc.; and (iii) A *program* part specialized to program reasoning. This system is referred to as a *global* system, since is it intended for reasoning about a program *as a whole*. In this survey, we focus on some useful proof rules from the program part, applicable to broad classes of properties. The reader is referred to [MP82] and [Pn86] for more detail.

The rules are presented in the form

A_1
.
.
.
A_n
——
B

where $A_1,\ldots,A_n$ are premises and B is the conclusion. The meaning is that if all the premises are shown to hold for a program then the conclusion is also true of the program.

The following *invariance* rule (INVAR) is adequate for proving most safety properties. Let ϕ be an assertion:

$$\frac{\begin{array}{l}\phi\\ G(\phi \Rightarrow X\phi)\end{array}}{G\phi}$$

Note that this rule really has the form of an induction rule. The first premise, the basis, ensures that ϕ holds initially. The second premise, the induction step, states that whenever ϕ holds, it also holds at the following moment. The conclusion is thus that ϕ always holds.

To perform the induction step, we must show that ϕ is preserved across all atomic actions of the program. In practice this can often be determined by inspection, considering only the potentially falsifying transitions and ignoring those which obviously cannot make ϕ *false*.

As an example, we now verify safety for Peterson's solution ([Pe81]) to the mutual exclusion problem shown in Figure 8. Each process has a noncritical section (l_0, m_0, resp.) in which it idles unless it needs access to its critical section (l_3, m_3, resp.), signalled by entry into its trying region (l_1 and l_2, m_1 and m_2, resp.) Presence in the critical sections should be mutually exclusive. The safety property we wish to establish is thus that the system never reaches a state where both processes are in their respective critical sections at the same time: $G(\neg(atl_3 \wedge atm_3))$.

It is helpful to establish several preliminary invariances. We use the notation $atl_{1\ldots3}$ to abbreviate $atl_1 \vee atl_2 \vee atl_3$:

$G\phi_1$,	ϕ_1 :	$y_1 \equiv atl_{1\ldots3}$
$G\psi_1$,	ψ_1 :	$y_2 \equiv atm_{1\ldots3}$
$G\phi_2$,	ϕ_2 :	$atl_3 \wedge atm_2 \Rightarrow t$
$G\psi_2$,	ψ_2 :	$atm_3 \wedge atl_2 \Rightarrow t$
$G\phi$,	ϕ :	$\neg(atl_3 \wedge atm_3)$

ϕ_1 plainly holds initially. Only transitions of process p_1 can affect it. Transitions $l_0 \rightarrow l_1$ leaves it true. Each of the other transitions of P_1 preserve its truth also, since y_1 is true whenever P_1 is at l_1, l_2, or l_3, and false when P_1 is at l_0. Thus $G\phi_1$ is established.

A similar argument proves $G\psi_1$.

ϕ_2 is vacuously true initially. The only potentially falsifying transitions for ϕ_2 are:

$l_1 \rightarrow l_2$ ensures atl_3 is *false* so ϕ_2 is preserved.

$l_2 \rightarrow l_3$ while atm_2—is enabled only when $\neg y_2 \vee t$ holds. Since y_2 is *true*, by virtue of ψ_1 and atm_2, it must be that t is *true* both before and after the transition. Hence ϕ_2 is preserved.

$m_1 \rightarrow m_2$ makes t *true* so that ϕ_2 is again preserved.

Thus $G\phi_2$ is established.

A similar argument establishes ψ_2.

Now to prove $G\phi$ we first note that ϕ holds initially. The only potentially falsifying transitions are in fact never enabled:

$l_2 \rightarrow l_3$ by process P_1 while process P_2 atm_3—By ψ_2, t is *false* and by ψ_1, y_2 holds. Since the enabling condition for the transition is $\neg y_2 \vee t$, the transition is never enabled.

$m_2 \rightarrow m_3$ by process P_2 while process P_1 atl_3—is similarly shown to be impossible.

Thus $G\phi$ (i.e, $G(\neg(atl_3 \wedge atm_3))$) is established.

We have the following liveness rule (LIVE), which is adequate for establishing eventualities based on a single step of a *helpful* process. Here we have formulae ϕ and ψ, and write $X_k p$ for $enabled_k \Rightarrow (executed_k \Rightarrow Xp)$, which means that the next execution of a step of process P_k will establish p. The rule is

$$\frac{\begin{array}{l} G(\phi \Rightarrow X(\phi \vee \psi)) \\ G(\phi \Rightarrow X_k\psi) \\ G(\phi \Rightarrow \psi \vee enabled_k) \end{array}}{G(\phi \Rightarrow F\psi)}$$

Often several invocations of LIVE must be linked together to prove an eventuality. We thus have the following rule, CHAIN:

$$\frac{G(\phi_i \Rightarrow F(\vee_{j<i}\phi_j \vee \psi))}{G(\vee_{i\leq k}\phi_i \Rightarrow F\psi)}$$

In many cases the rule CHAIN is adequate, in particular for finite state concurrent programs. In some instances, however, no a priori bound on the number of intermediate assertions ϕ_i can be given. We therefore use an assertion $\phi(a)$ with parameter a ranging over a given well-founded set $(W, <)$, which is a set W partially ordered by $<$ having no infinite decreasing sequence $a_1 > a_2 > a_3 > \ldots$. Note that this rule, WELL, generalizes the CHAIN rule, since we can take W to be the interval [1:k] with the usual ordering and $\phi(i) = \phi_i$.

$$\frac{G(\phi(a) \Rightarrow F(\exists b < a\ \phi(b) \vee \psi))}{G(\exists a\phi(a) \Rightarrow F\psi)}$$

We illustrate the application of the CHAIN rule on Peterson's [Pe81] algorithm for mutual exclusion. We wish to prove guaranteed accessibility:

$G(\mathrm{atl}_1 \Rightarrow F\mathrm{atl}_3)$

(which is sometimes also called absence of starvation for process P_1), indicating that whenever process 1 wants to enter its critical section, it will eventually be admitted.

We define the following assertions

ψ : atl_3
ϕ_1 : $\mathrm{atl}_2 \wedge \mathrm{atm}_2 \wedge \mathrm{t}$
ϕ_2 : $\mathrm{atl}_2 \wedge \mathrm{atm}_1$
ϕ_3 : $\mathrm{atl}_2 \wedge \mathrm{atm}_0$
ϕ_4 : $\mathrm{atl}_2 \wedge \mathrm{atm}_3$
ϕ_5 : $\mathrm{atl}_2 \wedge \mathrm{atm}_2 \wedge \neg\mathrm{t}$
ϕ_6 : atl_1

and establishing the corresponding temporal implication by an application of the LIVE rule in order to meet the the hypothesis of the CHAIN rule:

$G(\phi_6 \Rightarrow F(\phi_5 \vee \phi_4 \vee \phi_3 \vee \phi_2))$, using helpful process P_1
$G(\phi_5 \Rightarrow F\phi_4)$, using helpful process P_2
$G(\phi_4 \Rightarrow F\phi_3)$, using helpful process P_2
$G(\phi_3 \Rightarrow F\phi_2 \vee \psi)$, using helpful process P_1
$G(\phi_2 \Rightarrow F\phi_1 \vee \psi)$, using helpful process P_2
$G(\phi_1 \Rightarrow F\psi)$, using helpful process P_1

The CHAIN rule now yields $G(\phi_6 \Rightarrow F\psi)$, i.e., $G(\mathrm{atl}_1 \Rightarrow F\mathrm{atl}_3)$ as desired. The argument can be summarized in a *proof lattice* as depicted in Figure 9 (cf. [OL82], [MP82]).

7.3 Mechanical Synthesis of Concurrent Programs from Temporal Logic Specifications

One ambitious but promising possibility is that of automatically synthesizing concurrent programs from high-level specifications expressed in Temporal Logic. Here one deals with the *synchronization skeleton* of the program, which is an abstraction of the actual program where detail irrelevant to synchronization is suppressed. For example, in the synchronization skeleton for a solution to the critical section problem each process's critical section may be viewed as a single node since the internal structure of the critical section is unimportant. Most solutions to synchronization problems in the literature are in fact given as synchronization skeletons. Because synchronization skeletons are in general finite state, a propositional version of Temporal Logic suffices to specify their properties.

The synthesis method exploits the small model property of the propositional TL. It uses a decision procedure so that, given a TL formula, p, it will decide whether p is satisfiable or unsatisfiable. If p is satisfiable, a finite model of p is constructed. In this application, unsatisfiability of p means that the specification is inconsistent (and must be reformulated). If the formula p is satisfiable, then the specification it expresses is consistent. A model for p with a finite number of states is constructed by the decision procedure. The synchronization skeleton of a program meeting the specification can be read from this model. The small model property ensures that any program whose synchronization properties can be expressed in the TL can be realized by a system of concurrently running processes, each of which is a finite state machine.

One suitable logic is the branching time logic CTL. It has been used to specify and to synthesize, e.g., a starvation-free solution to the mutual exclusion problem (cf. [EC82]). Consider two processes P_1 and P_2, where each process is always in one of three regions of code: NCS_i—the *N*on*C*ritical *S*ection, TRY_i—the *TRY*ing Section, or CS_i—the *C*ritical *S*ection, which it cycles through, in order, repeatedly. When it is in region NCS_i, process P_i performs "noncritical" computations which can proceed in parallel with computations by other process P_j. At certain times, however, P_i may need to perform certain "critical" computations in the region CS_i. Thus, P_i remains in NCS_i as long as it has not yet decided to attempt critical section entry. When and if it decides to make this attempt, it moves into the region TRY_i. From there it enters CS_i as soon as possible, provided that the mutual exclusion constraint $\neg(atCS_1 \wedge atCS_2)$ is not violated. It remains in CS_i as long as necessary to perform its "critical" computations and then re-enters NCS_i.

It is assumed that only transitions between different regions of sequential code are recorded. Moves entirely within the same region are not considered in specifying synchronization. Moreover, the programs are running in a shared-memory environment with test-and-set primitives. The behavior of the system can be specified using the formulae listed below:

1. start state

   $atNCS_1 \wedge atNCS_2$

2. mutual exclusion

   $\mathrm{AG}\ (\neg(atCS_1 \wedge atCS_2))$

3. absence of starvation for P_i (i = 1,2)

   $\mathrm{AG}\ (atTRY_i \Rightarrow \mathrm{AF}\,atCS_i)$

plus some additional formulae to formally specify the information regarding the model of concurrent computation which was informally communicated in the above narrative. The global state transition diagram of a program meeting the conjunction of the above specifications, obtained by applying the synthesis method outlined, is shown in Figure 10. Solutions to other well known synchronization problems such as readers-writers and dining philophers can also be synthesized.

A closely related synthesis method for CSP programs based on the use of a decision procedure for PLTL was given in [MW84]. In the recent [PR89] a method for synthesizing an individual component of a reactive system from a specification in (essentially) CTL* is described. Earlier informal efforts toward synthesis of concurrent programs from TL-like formalisms include [La78] and [RK80].

There are a number of advantages to this type of automatic program synthesis method. It obviates the need to compose a program as well as the need to construct a correctness proof. Moreover, since it is algorithmic rather than heuristic in nature, it is both sound and complete. It is sound in that any program produced as a solution does in fact meet the specification. It is complete in that if the specification is satisfiable, a solution will be generated.

A drawback of this method is, of course, the (at least) exponential complexity of the decision procedure. Is this an insurmountable barrier to the development of this method into a practical software tool? Recall that while deciding satisfiability of propositional formulae requires exponential time in the worst case using the best known algorithms, the average case performance appears to be substantially better, and working automatic theorem provers and program verifiers are a reality. Similarly, the performance in practice of the decision procedure used by the synthesis method may be substantially better than the potentially exponential time worst case. (See [ESS89].) Furthermore, synchronization skeletons are generally small. It therefore seems conceivable that this approach may, in the long run, turn out to be useful in practical applications.

7.4 Automatic Verification of Finite State Concurrent Systems

The global state transition graph of a finite state concurrent system may be viewed as a finite temporal logic structure, and a model checking algorithm (cf. Section 6.3) can be applied to determine whether the structure is a model of a specification expressed as a formula in an appropriately chosen system of propositional TL. In other words, the model checking algorithm is used to determine whether a given finite state program meets a particular correctness specification. Provided that the model checking algorithm is efficient, this approach is potentially of wide applicability since a large class of concurrent programming problems have finite state solutions, and the interesting properties of many such systems can be specified in a propositional TL. For example, many network communication protocols can be modeled as a finite state system.

The basic idea behind this mechanical model checking approach to verification of finite state systems is to make brute force graph reachability analysis efficient and expressive through the use of TL as an assertion language. Of course, research in protocol verification has attempted to exploit the fact that protocols are frequently finite state, making exhaustive graph reachability analysis possible. The advantage offered by model checking seems to be that it provides greater flexibility in formulating specifications through the use of TL as a single, uniform assertion language that can express a wide variety of correctness properties. This makes it possible to reason about, e.g., both safety and liveness properties with equal facility.

Historically, [Pn77] showed that the problem of deciding truth of a linear temporal formula over a finite structure was decidable. However, his decision procedure was nonelementary, and the problem is PSPACE-complete in general (Theorem 6.17). The term "model checking" was coined by [CE81], who gave an efficient (polynomial time) model checking algorithm for the branching time logic CTL, and first proposed that it could be used as the basis of a practical automatic verification technique. At roughly the same time, [QS82] gave a model checking algorithm for a similar branching time logic, but did not analyze its complexity.

To illustrate how model checking algorithms work, we now describe a simple model checking algorithm for CTL. Note that it is similar to the global flow analysis algorithms used in compiler optimization. Assume that M = (S,R,L) is a finite structure and p_0 is a CTL formula. The goal is to determine at which states s of M, we have M,s $\models$ p_0. The algorithm is designed to operate in stages: the 1st stage processes all subformulae of p_0 of length 1, the 2nd stage processes all subformulae of p_0 of length 2, and so on. At the end of the ith stage, each state will be labeled with the set of all subformulae of length $\leq$ i that are true at the state. To perform the labelling at stage i, information gathered in earlier stages is used. For example, subformulae q $\wedge$ r should be placed in the label of a state s precisely when q and r are both already in the label of s. For the modal subformula A[q U r], information from the successor states of s, as well as state s itself, is used. Since A[q U r] $\equiv$ q $\vee$ AXA[q U r], A[q U r] is initially added to the label of each state already labelled with r. Then satisfaction of A[q U r] is propagated outward, by repeatedly adding A[q U r] to the label of each state labelled by q and having A[q U r] in the label of all successors.

Let

$$(A[q\ U\ r])^1 = r$$
$$(A[q\ U\ r])^{j+1} = r \vee (q \wedge AX(A[q\ U\ r])^j)$$

It can be shown that M,s $\models$ $(A[q\ U\ r])^j$ iff M,x $\models$ A[q U r] and along every path starting at s, r holds within distance j. Thus, states where $(A[q\ U\ r])^1$ holds are found first, then states where $(A[q\ U\ r])^2$ holds, etc. If A[q U r] holds, then $(A[q\ U\ r])^{card(S)}$ must hold since all loop-free paths in M are of length $\leq$ card(S). Thus, if after card(S) steps of propagating outward, A[q U r] has still not been found to hold at state s, then A[q U r] is false at s. Satisfaction of the other CTL modality E[p U q] propagates outward in the same fashion.

This version of the algorithm can be naively implemented to run in time linear in the length of p_0 and quadratic in the size of structure M. A more clever version of the algorithm can be implemented to run in time linear in the length of the input formula p and the size of M (cf. [CES86]).

```
for i = 1 to length($p_0$)
    for each subformula p of $p_0$ of length i
        Case on the form of p
            p = P, an atomic proposition /* nothing to do */

            p = $q \wedge r$ : for each $s \in S$
                            if $q \in L(s)$ and $r \in L(s)$ then
                                add $q \wedge r$ to L(s)
                        end
            p = $\neg q$ :   for each $s \in S$
                            if $q \notin L(s)$ then
                                add $\neg q$ to L(s)
                        end
            p = EXq:  for each $s \in S$
                            if (for some successor t of s, $q \in L(t)$) then
                                add EXq to L(s)
                        end
            p = A[q U r] : for each $s \in S$
                                if $r \in L(s)$ then
                                    add A[q U r] to L(s)
                            end
                        for j = 1 to Card(S)
                            for each $s \in S$
                                if $q \in L(s)$ and (for each successor t of s,
                                    A[q U r]$\in$L(t)) then add A[q U r] to L(s)
                            end
                        end
            p = E[q U r]: for each $s \in S$
                                if $r \in L(s)$ then
                                    add E[q U r] to L(s)
                            end
                for j = 1 to Card(s)
                            for each $s \in S$
                                if $q \in L(s)$ and (for some successor t of s,
                                    E[q U r] $\in$ L(t)) then add E[q U r] to L(s)
                            end
                end
        end of case
    end
end
```

One limitation of the logic CTL is, of course, that it cannot express correctness under fair scheduling assumptions. However, the extended logic FairCTL described in [EL85] can express correctness under fairness (cf. [QS83]). An FCTL specification (p_0, Φ_0) consists of a functional assertion p_0, which is a state formula, and an underlying fairness assumption Φ_0, which is a pure path formula. The functional assertion p_0 is expressed in essentially CTL syntax with basic modalities of the form either A_Φ ("for all *fair* paths") or E_Φ ("for some *fair* path") followed by one of the linear time operators F, G, X, or U. The path quantifiers range over paths meeting the fairness constraint

Φ_0, which is a boolean combination of the infinitary linear time operators $\overset{\infty}{F}$ ("infinitely often") and $\overset{\infty}{G}$ ("almost always"), applied to propositional arguments. We can then view a subformula such as $A_\Phi FP$ of functional assertion p_0 as an abbreviation for the CTL* formula $A[\Phi_0 \Rightarrow FP]$. Similarly, $E_\Phi GP$ abbreviates $E[\Phi_0 \wedge GP]$. In this way FairCTL inherits its semantics from CTL*. Provided that Φ_0 is in the canonical form

$$\vee_{i=1}^{n} \wedge_{j=1}^{n_i} (\overset{\infty}{F}p_{ij} \vee \overset{\infty}{F}q_{ij})$$

then the model checking problem for FairCTL can be solved in time that is linear in the input structure size and small polynomial in the specification size.

Nevertheless, there are still correctness properties that one might like to describe that are not expressible within FairCTL, although they are describable in CTL* or even PLTL. The PSPACE-completeness of these latter logics, on first hearing, would seem to be a serious drawback. Lichtenstein and Pnueli [LP85] noted, however, that model checking is a problem with two input parameters, the structure and the specification, and then proceeded to develop a model checking algorithm for PLTL of complexity exponential in the the length of the specification but only linear in the size of the structure. They argued that since specifications are generally quite short while the structures representing programs are usually quite large, the exponential complexity in the specification size can be discounted. In practice, the dominating factor in the complexity should thus be the linear growth in the structure size.

It is worth pointing out that model checking, despite (because of?) its simplicity, is one approach to automatic verification that really seems to be useful in practice. It has been used to verify a large variety of finite state concurrent programs. These programs range from examples in the academic literature on concurrency to large-scale network communication protocols. For instance, a solution to the mutual exclusion problem given in [OL82] and proved correct there manually using linear TL is actually finite state. It was mechanically verified using the CTL model checking algorithm as described in [CES83]. Model checking is also applicable to the design of VLSI hardware and asynchronous circuits: Clarke has developed an efficient implementation of the CTL model checker along with various pieces of support software, which together forms the EMC (Extended Model Checker) system at CMU. In [MC86] the use of the EMC system resulted in the detection of a previously unknown error in a circuit for a self-time queue element published in the text [MC78]. Other applications to the design of sequential circuits are discussed in [BCD85], [BCDM86a], and [DC86], as well as the overview article [CG87]. Finally, model checking is applicable to large-scale network communication protocols. Indeed, one project in France [Si87] has bought dedicated hardware to use for model checking network protocols. Finite state systems with on the order of 10^5 states (and arcs) can currently be handled.

Despite the above practical successes, a potentially serious drawback to the entire model checking approach is that the size of the global state transition graph grows exponentially with the number of processes. Recent work in [CG86], [SG87], [CG87] suggests that it may be possible to avoid this exponential blowup in some cases for concurrent systems with many "copies" of the same process, although this is not possible in general (cf. [AK86]). Other work on reducing the size of the state graph based on hierarchical specification and hiding of states at lower levels of abstraction is presented in [MC85].

8 Other Modal and Temporal Logics in Computer Science

8.1 Classical Modal Logic

The class of Modal Logics was originally developed by philosophers to study different "modes" of truth. Such modes include possibility, necessity, obligation, knowledge, belief, and perception. Among the most important modes of truth are what "must be" true (necessity) and what "may be" true (possibility). For example, the assertion P may be false in the present world, and yet the assertion *possibly* P true, if there exists another world where P is true. The assertion *necessarily* P is true provided that P is true in all worlds.

Thus we have the well known *possible worlds* semantics of Kripke, where the truth value of a modal assertion at a world depends on the truth value(s) of its subassertions(s) at other possible worlds. This is formalized in terms of a *Kripke structure* M = (S,R,L) consisting of an underlying set S of possible *worlds*, also called *states*, an accessibility relation $R \subseteq S \times S$ between worlds, and a labelling L which provides an interpretation of primitive (i.e, nonmodal) assertions at each world. The technical definitions are such that *possibly* P is true at world s iff P is true at some world accessible from s, and *necessarily* P is true at world s iff P is true at all worlds accessible from s.

As we have seen, Temporal Logic is a particular kind of Modal Logic that has been specialized for reasoning about program behavior. Temporal Logic provides a much richer set of modalities, varying in how their truth value depends on which argument(s) hold(s) at which worlds, with the accessibility relation corresponding to the evolution of a concurrent system over time.

8.2 Propositional Dynamic Logic

An alternative development of a modal logic framework for program reasoning is represented by Dynamic Logic, originally proposed by Pratt [Pr76] in the first order version, specialized to the propositional version by Fischer and Ladner [FL79], and, in general studied intensively by Harel [Ha79] and others. (Detailed treatements of Dynamic logic can be found in [KT89] and [Ha84].) The basic modalities of Propositional Dynamic Logic (PDL) are of the form $<\alpha>$p where α is a regular expression over "atomic programs" and p is a formula. The intuitive meaning is that there exists an execution of program α leading from the present state to a state where p holds. PDL may be viewed as a propositional Branching Time Logic, with basic modalities of the form Eβ, where β is a regular expression over atomic propositions (node labels) and atomic programs (arc labels), and which means that there exists a path (a sequence of alternating states and arcs) starting at the present state that matches the regular expression β. A variety of extensions of PDL have been proposed in order to increase its expressive power. One is of particular interest to Temporal Logicians, viz., that with a repetition construct referred to a PDL + repeat or Δ-PDL. In Temporal Logic terms, its basic modalities are of the form Eβ, where β is an ω-regular expression such as $\alpha\gamma^{\omega}$. The ω iteration operator corresponds to the repeat operator Δ. Δ-PDL strictly subsumes CTL* in expressive power, and is thus able to express modalities such as AFp that cannot be expressed in ordinary PDL. Historically, Δ-PDL is important for reasons beyond its high expressive power. It is with Δ-PDL that automata-theoretic techniques for testing satisfiability were pioneered by Streett [St81].

8.3 Probabilistic Logics

Various probabilistic Temporal Logics have been proposed. For instance, [Lehmann & Shelah] describe logic, TC, with essentially the same syntax as CTL*, but where Ap means for almost all paths (i.e., for a set of paths of measure 1) p holds, and Ep means for significantly many paths (i.e, for a set of paths of positive measure) p holds. They give a deterministic double exponential time decision procedure and a sound and complete axiomitization for it. Interestingly, the logic TC has the same axiomitization as the logic MPL (Modal Process Logic) of Abrahamson [Ab80]. MPL has essentially the same syntax as CTL* but interprets it over more abstract structures, where the set of paths is not required to be generated by a binary relation. A probabalistic version of CTL is considered in [HS84].

8.4 Fixpoint Logics

Temporal operators can be characterized in terms of extremal fixpoints of monotonic functionals. Let M = (S,R,L) be a structure. We use PRED(S) to denote the lattice of total predicates over state set S, where each predicate is identified with the set of states which make it true and the ordering on predicates is set inclusion. Then a formula p defines a member of PRED(S), $\{ s \in S: M,s \models p\}$, and if it contains an atomic proposition Q, e.g., p(Q), it defines a function PRED(S) $\rightarrow$ PRED(S) where the value of p(Q) varies as Q varies. Let τ : PRED(S) $\rightarrow$ PRED(S) be given; then

- τ is said to be *monotonic* provided $P \subseteq Q$ implies $\tau(P) \subseteq \tau(Q)$
- τ is said to be $\cup$-*continuous* provided that $P_1 \subseteq P_2 \subseteq P_3$... implies $\tau(\cup_i P_i) = \cup_i \tau(P_i)$.
- τ is said to be $\cap$-*continuous* provided that $P_1 \supseteq P_2 \supseteq P_3$... implies $\tau(\cap_i P_i) = \cap_i \tau(P_i)$.

A predicate P is said to be a *fixpoint* of functional τ if $P = \tau(P)$. The theorem of Tarski-Knaster ([Ta55]) ensures that a monotonic functional τ: PRED(S) $\rightarrow$ PRED(S) always has a least fixpoint, $\mu Z.\tau(Z) = \cap\{ Y: \tau(Y) = Y\}$, and a greatest fixpoint $\nu Z.\tau(Z) = \cup\{ Y: \tau(Y) = Y\}$. Whenever τ is $\cup$-continuous then $\mu Z.\tau(Z) = \cup_i \tau^i($*false*, and whenever τ is $\cap$-continuous then $\nu Z.\tau(Z) = \cap_i \tau^i($*true*$)$. (Note: $\tau^2($*false*$) = \tau(\tau($*false*$))$, etc.)

For example, shown below are the fixpoint characterizations for certain CTL modalities.

$$\begin{array}{ll} EFP = \mu Z.P \vee EXZ & AGP = \nu Z.P \wedge AXZ \\ AFP = \mu Z.P \vee AXZ & EGP = \nu Z.P \wedge EXZ \end{array}$$

Intuitively, the properties characterized as least fixpoints correspond to eventualities, while those characterized as greatest fixpoints are invariance properties

Assume for simplicity that each state in the underlying structure has a finite number of successors. Then each of the above functionals is both $\cup$-continuous and $\cap$-continuous in the argument Z, and we can readily establish the correctness for the above characterizations. For example, to show that $EFP = \mu Z.\tau(Z)$, with $\tau(Z) = P \vee EXZ$, it suffices to show that for each i (ranging over

|N), $\tau^i(\mathit{false})$ = { states s in M : there exists a path of length i in M from state s to some state t such that M,t $\models$ P}

These fixpoint characterizations are used in the model checking algorithm of section 7 and in the tableau-based decision procedure of section 6.

We can define an entire logic built-up from atomic proposition constants P, Q,..., atomic proposition variables Y,Z,... boolean connectives $\wedge$, $\vee$, $\neg$, the nextime operators EX, AX, and the least fixpoint μ and greatest fixpoint ν operators. We require that each formula be syntactically monotone, meaning that fixpoint formulae such as $\mu Z.\tau(Z)$ (or $\nu X.\tau(Z)$) are legal only when Z appears under an even number of negations within τ. The semantics is given in the obvious way suggested above.

Essentially this system was dubbed the "the Propositional Mu-Calculus" by Kozen [Ko83]. This Mu-Calculus has very considerable expressive power. It can encode (and in fact subsumes) CTL, FairCTL, CTL*, and, interpreted over multi-process structures, also PDL and PDL+repeat. It practical terms it also allows expression of extended modalities such as P is true at all even moments along all futures, which is captured by $\nu Z.P \wedge AXAXZ$. Related systems were considered in [EC80] and [PR81]. Other proposals for formalisms based on fixpoints can be found in, e.g., [deBS69], [Pa70], [deRo76], [Di76], and [Pa80].

8.5 Knowledge

There has recently been interest in the development of modal and temporal logics for reasoning about the states of knowledge in reactive systems. Knowledge can be especially important in the realm of distributed systems where processes are geographically dispersed and, at any given moment, possess only incomplete knowledge regarding the status of other processes in the system. Indeed, in many informal instances of reasoning about the behavior of distributed systems, it is a natural metaphor to refer to what a process *knows*. Logics of knowledge represent an effort to provide a formal basis for such reasoning.

A number of systems have been proposed (cf. [HM84], [Le84], [LR86], [DM86]). Typical modalities include K_ip which means that "process i knows p" and Cp which means that "p is common knowledge" in the sense that "all processes know p, all processes know that all processes know p, all processes know that all process know that all process know p, ..." These modalites of knowledge can be combined in various ways with temporal operators to permit reasoning about distributed systems. Certain semantic constraints, expressed as axioms (e.g. $K_i Xp \Rightarrow XK_i p$), are usually required. Subtle interactions between the syntax of the logic and the assumptions made regarding the model of distributed computation can lead to widely varying complexities for the decision problems of the resulting logics (cf. [HV86]). In general, this seems a promising area for future research. We refer the reader to the excellent survey by Halpern [Ha87] for an in-depth treatment.

Acknowledgement. We would like to thank Amir Pnueli for very helpful comments.

9 Bibliography

[AB80] Abrahamson, K., Decidability and Expressiveness of Logics of Processes, PhD Thesis, University of Washington, 1980.

[AK86] Apt, K. and Kozen, D., Limits for Automatic Verification of Finite State Systes, IPL vol. 22, no. 6., pp. 307-309, 1986.

[AS85] Alpern, B. and Schneider, F., Defining Safety and Liveness, Tech. Report, Cornell Univ., 1985.

[deBS69] de Bakker, J. and Scott, D., A Theory of Programs, Unpublished notes, IBM Seminar, Vienna, 1969.

[BKP84] Barringer, H., Kuiper, R., and Pnueli, A., Now You May Compose Temporal Logic Specifications, STOC84.

[BKP86] Barringer, H., Kuiper, R., and Pnueli, A., A Really Abstract Concurrent Model and its Temporal Logic, pp. 173-183, POPL86.

[BHP82] Ben-Ari, M, Halpern, J. Y., and Pnueli, A., Deterministic Propositional Dynamic Logic: Finite Models, Complexity, and Completeness, JCSS, v. 25, pp. 402-417, 1982

[BPM81] Ben-Ari, M., Pnueli, A. and Manna, Z. The Temporal Logic of Branching Time. ACM POPL 81; jounral version in Acta Informatica vol. 20, pp. 207-226, 1983.

[BCD85] Browne, M., Clarke, E. M., and Dill, D. Checking the Correctness of sequential Circuits, Proc. 1985 IEEE INt. Conf. Comput. Design, Port Chester, NY pp. 545-548

[BCDM86a] Browne, M., Clarke, E. M., and Dill, D., and Mishra, B., Automatic verification of sequential circuits using temporal logic, IEEE Trans. Comp. C-35(12), pp. 1035-1044, 1986

[Bu74] Burstall, R., Program Proving Considered as Hand Simulation plus Induction, IFIP Congress, Amsterdam, pp. 308-312, 1974.

[CE81] Clarke, E. M., Emerson, E.A., Design and Synthesis of Synchronization Skeletons Using Branching Time Temporal Logic, IBM Logics of Programs Workshop, Springer LNCS #131, pp. 52-71, May 1981.

[CES83] Clarke, E. M., Emerson, E. A., and Sistla, A. P., Automatic Verification of Finite State Concurrent System Using Temporal Logic, 10th ACM Symp. on Principles of Prog. Lang., Jan. 83; journal version appears in *ACM Trans. on Prog. Lang. and Sys.*, vol. 8, no. 2, pp. 244-263, April 1986.

[CG86] Clarke, E. M., Grumberg, O. and Browne, M.C., Reasoning about Networks with Many Identical Finite State Processes, Proc. 5th ACM PODC, pp. 240-248, 1986.

[CG87] Clarke, E. M. and Grumberg, O., Avoiding the State Explosion Problem In Temporal Model Checking, PODC87.

[CG87b] Clarke, E. M. and Grumberg, O. Research on Automatic Verification of Finite State Concurrent Systems, Annual Reviews in Computer Science, 2, pp. 269-290, 1987

[CM83] Clarke, E. M., Mishra, B., Automatic Verification of Asynchronous Circuits, CMU Logics of Programs Workshop, Springer LNCS #164, pp. 101-115, May 1983.

[CVW85] Courcoubetis, C., Vardi, M. Y., and Wolper, P. L., Reasoning about Fair Concurrent Programs, Proc. 18th STOC, Berkeley, Cal., pp. 283-294, May 86.

[Di76] Dijkstra, E. W. , A Discipline of Programming, Prentice-Hall, 1976.

[[DC86] Dill, D. and Clarke, E.M., Automatic Verification of Asynchronous Circuits using Temporal Logic, IEE Proc. 133, Pt. E 5, pp. 276-282, 1986.

[DM86] Dwork, C. and Moses, Y. Knowldge and Common Knowledge in a Byzantine Environment I: Crash Failures. In Proc. of the 1st Conf. on Theoretical Aspects of Reasoning about Knowledge, J. Y. Halpern ed., Los Altos, Cal., Morgan Kaufmann, pp 149-170, 1986.

[Em83] Emerson, E. A., Alternative Semantics for Temporal Logics, Theor. Comp. Sci., v. 26, pp. 121-130, 1983.

[EC80] Emerson, E. A., and Clarke, E. M., Characterizing Correctness Properties of Parallel Programs as Fixpoints. Proc. 7th Int. Colloquium on Automata, Languages, and Programming, Lecture Notes in Computer Science #85, Springer-Verlag, 1981.

[EC82] Emerson, E. A., and Clarke, E. M., Using Branching Time Temporal Logic to Synthesize Synchronization Skeletons, *Science of Computer Programming*, vol. 2, pp. 241-266, Dec. 1982.

[EH85] Emerson, E. A., and Halpern, J. Y., Decision Procedures and Expressiveness in the Temporal Logic of Branching Time, *Journal of Computer and System Sciences*, vol. 30, no. 1, pp. 1-24, Feb. 85.

[EH86] Emerson, E. A., and Halpern, J. Y., 'Sometimes' and 'Not Never' Revisited: On Branching versus Linear Time Temporal Logic, *JACM*, vol. 33, no. 1, pp. 151-178, Jan. 86.

[EL85] Emerson, E. A. and Lei, C. L., Modalities for Model Checking: Branching Time Strikes Back, pp. 84-96, ACM POPL85; journal version appears in Sci. Comp. Prog. vol. 8, pp 275-306, 1987.

[ES84] Emerson, E. A., and Sistla, A. P., Deciding Full Branching Time Logic, *Information & Control*, vol. 61, no. 3, pp. 175-201, June 1984.

[ESS89] Emerson, E. A., Sadler, T. H. , and Srinivasan, J. Efficient Temporal Reasoning, pp 166-178, 16th ACM POPL, 1989.

[En72] Enderton, H. B., A Mathematical Introduction to Logic, Academic Press, 1972.

[FL79] Fischer, M. J., and Ladner, R. E, Propositional Dynamic Logic of Regular Programs, JCSS vol. 18, pp. 194-211, 1979.

[Fr86] Francez, N., Fairness, Springer-Verlag, NY, 1986.

[FK84] Francez, N., and Kozen, D., Generalized Fair Termination, 11th Annual ACM Symp. on Principles of Programming Languages, 1984, pp. 46-53.

[GPSS80] Gabbay, D., Pnueli A., Shelah, S., Stavi, J., On The Temporal Analysis of Fairness, 7th Annual ACM Symp. on Principles of Programming Languages, 1980, pp. 163-173.

[Ha87] Halpern, J. Y., Using Reasoning about Knowledge to Analyze Distributed Systems, Annual Reviews in Computer Science, 2, pp. 37-68, 1987.

[HM84] Halpern, J. Y. and Moses, Y. Knowledge and Common Knowledge in a Distributed Environment, Proc. 3rd ACM Symp. PODC, pp. 50-61.

[Ha86] Halpern, J. Y. and Shoham, Y., A Propositional Modal Logic of Time Intervals, IEEE LICS, pp. 279-292, 1986.

[Ha79] Harel, D., Dynamic Logic: Axiomatics and Expressive Power, PhD Thesis, MIT, 1979; also available in Springer LNCS Series no. 68, 1979.

[HA84] Harel, D., Dynamic Logic, in Handbook of Philosophical Logic vol. II: Extensions of Classical Logic, ed. D. Gabbay and F. Guenthner, D. Reidel Press, Boston, 1984, pp. 497-604. Applications, 16th STOC, pp. 418-427, May 84.

[HS84] Hart, S. and Sharir, M. Probabalistic Temporal Logics for Finite and Bounded Models, 16th ACM STOC, pp. 1-13, 1984.

[Ho78] Hoare, C. A. R., Communicating Sequential Processes, CACM, vol. 21, no. 8, pp. 666-676, 1978.

[Ha82] Hailpern, B., Verifying Concurrent Processes Using Temporal Logic, Springer-Verlag LNCS no. 129, 1982.

[HO80] Hailpern, B. T., and Owicki, S. S., Verifying Network Protocols Using Temporal Logic, In Proceedings Trends and Applications 1980: Computer Network Protocols, IEEE Computer Society, 1980, pp. 18-28.

[HR74] Hossley, R., and Rackoff, C, The Emptiness Problem For Automata on Infinite Trees, Proc. 13th IEEE Symp. Switching and Automata Theory, pp. 121-124, 1972.

[HS84] Hart, S., and Sharir, M., Probabilistic Temporal Logics for Finite and Bounded Models, 16th STOC, pp. 1-13, 1984.

[HT87] Hafer, T., and Thomas, W., Computation Tree Logic CTL* and Path Quantifiers in the Monadic Theory of the Binary Tree, ICALP87.

[Ka68] Kamp, Hans, Tense Logic and the Theory of Linear Order, PhD Dissertation, UCLA 1968.

[Ko87] Koymans, R., Specifying Message Buffers Requires Extending Temporal Logic, PODC87.

[Ko83] Kozen, D., Results on the Propositionnal Mu-Calculus, Theor. Comp. Sci., pp. 333-354, Dec. 83

[KP81] Kozen, D. and Parikh, R. An Elementary Proof of Completeness for PDL, Theor. Comp. Sci., v. 14, pp. 113-118, 1981

[KT87] Kozen, D. and Tiuryn, J., Logics of Programs, this volume

[Ku86] Kurshan, R. P. , Testing containment of omega regular languages, Tech. Report 1121-861010-33-TM, ATT Bell Labs, Murray Hill, NJ, 1986.

[LR86] Ladner, R. and Reif, J. The Logic of Distributed Protocols, in Proc. of Conf. On Theor. Aspects of reasoning about Knowledge, ed. J Halpern, pp. 207-222, Los Altos, Cal., Morgan Kaufmann

[LA80] Lamport, L., Sometimes is Sometimes "Not Never"—on the temporal logic of programs, 7th Annual ACM Symp. on Principles of Programming Languages, 1980, pp. 174-185.

[La83] Lamport, L., What Good is Temporal Logic?, Proc. IFIP, pp. 657-668, 1983.

[La78] Laventhal, M. Synthesis of Synchronization Code for Data Abstractions, PhD Thesis, MIT, 1978.

[Le84] Lehmann, D., Knowledge, Common Knowldge, and Related Puzzles, 3rd ACM Symp. PODC 84, pp 62-67.

[LPS81] Lehmann. D., Pnueli, A., and Stavi, J., Impartiality, Justice and Fairness: The Ethics of Concurrent Termination, ICALP 1981, LNCS Vol. 115, pp 264-277.

[LS82] Lehmann, D., and Shelah, S. Reasoning about Time and Chance, Inf. and Control, vol. 53, no. 3, pp. 165-198, 1982.

[LP85] Lichtenstein, O. and Pnueli, A., Checking that Finite State Concurrent Programs Satisfy their Linear Specification, POPL85, pp. 97-107, Jan. 85.

[LPZ85] Lichtenstein, O, Pnueli, A. ,and Zuck, L. The Glory of the Past, Brooklyn College Conference on Logics of Programs, Springer-Verlag LNCS, June 1985.

[MP82a] Manna, Z., and Pnueli, A., Verification of Concurrent Programs: The Temporal Framework, in The Correctness Problem in Computer Science, Boyer & Moore (eds.), Academic Press, pp. 215-273, 1982.

[MP81] Manna, Z. and Pnueli, A., Verification of Concurrent Programs: Temporal Proof Principles, in Proc. of Workshop on Logics of Programs, D. Kozen (ed.), Springer LNCS #131, pp. 200-252, 1981.

[MP82] Manna, Z. and Pnueli, A., Verification of Concurrent Programs: A Temporal Proof System, Proc. 4th School on Advanced Programming, Amsterdam, The Netherlands, June 82.

[MP83] Manna, Z. and Pnueli, A., How to Cook a Proof System for your Pet Language, ACM Symp. on Princ. of Prog. Languages, pp. 141-154, 1983.

[MP84] Manna, Z. and Pnueli, A., Adequate Proof Principles for Invariance and Liveness Properties of Concurrent Programs, Science of Computer Programming, vol. 4, no. 3, pp. 257-290, 1984.

[MP87a] Manna, Z. and Pnueli, A. Specification and Verification of Concurrent Programs by $\forall$-automata, Proc. 14th ACM POPL, 1987

[MP87b] Manna, Z. and Pnueli, A. A Hierarchy of Temporal Properties, PODC7.

[MW78] Manna, Z., and Waldinger, R., Is "sometimes" sometimes better than "always"?: Intermittent assertions in proving program correctness, CACM, vol. 21, no. 2, pp. 159-172, Feb. 78

[MW84] Manna, Z. and Wolper, P. L., Synthesis of Communicating Processes from Temporal Logic Specifications, vol. 6, no. 1, pp. 68-93, Jan. 84.

[McN66] McNaughton, R., Testing and Generating Infinite Sequences by a Finite Automaton, Information and Control, Vol. 9, 1966.

[MP62] McNaughton, R. and Pappert, S. Counter-Free automata, MIT Press, 1962.

[MC80] Mead, C. and Conway, L., Introduction to VLSI Systems, Addison-Wesley, Reading, Mass., 1980.

[Me74] Meyer, A. R., Weak Monadic Second Order Theory of One Successor is Not Elementarily Recursive, Boston Logic Colloquium, Springer-Verlag Lecture Notes in Math. no. 453, Berlin/New York, 1974.

[Mo83] Moszkowski, B., Reasoning about Digital Circuits, PhD Thesis, Stanford Univ, 1983.

[MU63] Muller, D. E., Infinite Sequences and Finite Machines, 4th Ann. IEEE Symp. of Switching Theory and Logical Design, pp. 3-16, 1963.

[NDOG86] Nguyen, V., Demers, A., Owicki, S., and Gries, D., A Model and Temporal Proof System for Networks of Processes, Distr. Computing, vol. 1, no. 1, pp 7-25, 1986

[OL82] Owicki, S. S., and Lamport, L., Proving Liveness Properties of Concurrent Programs, ACM Trans. on Programming Languages and Syst., Vol. 4, No. 3, July 1982, pp. 455-495.

[Pa78] Parikh, R., A Decidability Result for Second Order Process Logic, Proc. 19th IEEE FOCS, pp. 177-183, 1978.

[Pa70] Park, D., Fixpoint Induction and Proof of Program Semantics, in Machine Intelligence, eds. B. Meltzer and D. Michie, vol. 5, pp. 59-78, Edinburgh Univ. Press, Edinburgh, 1970.

[PA80] Park, D., On The Semantics of Fair Parallelism, Abstract Software Specification, LNCS Vol. 86, Springer Verlag, 1980, pp. 504-524.

[PW84] Pinter, S., and Wolper, P. L., A Temporal Logic for Reasoning about Partially Ordered Computations, Proc. 3rd ACM PODC, pp. 28-37, Vancouver, Aug. 84

[Pe81] Peterson, G. L., Myths about the Mutual Exclusion Problem, Inform. Process. Letters, vol. 12, no. 3, pp. 115-116, 1981.

[PN77] Pnueli, A., The Temporal Logic of Programs, 18th annual IEEE-CS Symp. on Foundations of Computer Science, pp. 46-57, 1977.

[Pn81] Pnueli, A., The Temporal Semantics of Concurrent Programs, Theor. Comp. Sci., vol. 13, pp 45-60, 1981.

[PN83] Pnueli, A., On The Extremely Fair Termination of Probabilistic Algorithms, 15 Annual ACM Symp. on Theory of Computing, 1983, 278-290.

[Pn84] Pnueli, A., In Transition from Global to Modular Reasoning about Concurrent Programs, in Logics and Models of Concurrent Systems, ed. K. R. Apt, Springer, 1984.

[Pn85] Pnueli, A., Linear and Branching Structures in the Semantics and Logics of Reactive Systems, Proceedings of the 12th ICALP, pp. 15-32, 1985.

[Pn86] Pnueli, A., Applications of Temporal Logic to the Specification and Verification of Reactive Systems: A Survey of Current Trends, in Current Trends in Concurrency: Overviews and Tutorials, ed. J. W. de Bakker, W.P. de Roever, and G. Rozenberg, Springer LNCS no. 224, 1986.

[PR76] Pratt, V., Semantical Considerations on Floyd-Hoare Logic, 17th FOCS, pp. 109-121, 1976.

[Pr79] Pratt, V., Models of Program Logics, Proc. 20th IEEE FOCS, pp. 115-122, 1979.

[PR81] Pratt, V., A Decidable Mu-Calculus, 22nd FOCS, pp. 421-427, 1981.

[Pr] Prior, A., Past, Present, and Future, Oxford Press.

[RU71] Rescher, N., and Urquhart, A., Temporal Logic, Springer-Verlag, 1971.

[QS82] Queille, J. P., and Sifakis, J., Specification and verification of concurrent programs in CESAR, Proc. 5th Int. Symp. Prog., Springer LNCS no. 137, pp. 195-220, 1982.

[QS83] Queille, J. P., and Sifakis, J., Fairness and Related Properties in Transition Systems, Acta Informatica, vol. 19, pp. 195-220, 1983.

[RK80] Ramarithram, K., and Keller, R., Specification and Synthesis of Synchronizers, Porc. 9th Int. Conf. on Par. Processing, pp. 311-321, 1980.

[deR76] de Roever, W. P., Recursive Program Schemes: Semantics and Proof Theory, Math. Centre Tracts no. 70, Amsterdam, 1976.

[Sa88] Safra, S., On the complexity of omega-automata, Proc. 29th IEEE FOCS, pp. 319-327, 1988.

[Si83] Sistla, A. P., Theoretical Issues in the Design of Distributed and Concurrent Systems, PhD Thesis, Harvard Univ., 1983.

[SC85] Sistla, A. P., and Clarke, E. M., The Complexity of Propositional Linear Temporal Logic, J. ACM, Vol. 32, No. 3, pp.733-749.

[SCFM84] Sistla, A. P., Clarke, E. M., Francez, N., and Meyer, A. R., Can Message Buffers be Axiomitized in Temporal Logic?, Information & Control, vol. 63., nos. 1/2, Oct./Nov. 84, pp. 88-112.

[SG87] Sistla, A. P., and German, S. M., Reasoning about Many Processes, LICS87.

[Si85] Sistla, A. P., Characterization of Safety and Liveness Properties in Temporal Logic, PODC85.

[SVW87] Sistla, A. P., Vardi, M. Y., and Wolper, P. L., The Complementation Problem for Buchi Automata with Applications to Temporal Logic, Theor. Comp. Sci., v. 49, pp 217-237, 1987.

[Si87] Sifakis, J., Personal Communication, April 87.

[SMS82] Schwartz, R., and Melliar-Smith, P. From State Machines to Temporal Logic: Specification Methods for Protocol Standards, IEEE Trans. on Communication, COM-30, 12, pp. 2486-2496, 1982.

[SMV83] Schwartz, R., Melliar-Smith, P. and Vogt, F. An Interval Logic for Higher-Level Temporal Reasoning, Proc. 2nd ACM PODC, Montreal, pp. 173-186, Aug. 83.

[ST82] Streett, R., Propositional Dynamic Logic of Looping and Converse, Information and Control 54, 121-141, 1982. (Full version: Propositional Dynamic Logic of Looping and Converse, PhD Thesis, MIT Lab for Computer Science, 1981.)

[SE84] Streett, R., and Emerson, E. A., The Propositional Mu-Calculus is Elementary, ICALP84, pp 465 -472, July 84.

[Ta55] Tarksi, A., A Lattice-Theoretical Fixpoint Theorem and its Applications, Pacific. J. Math., 55, pp. 285-309, 1955.

[Th79] Thomas, W., Star-free regular sets of omega-sequences, Information and Control, v. 42, pp. 148-156, 1979

[Th89] Thomas, W., Automata on Infinite objects, this volume

[Va85] Vardi, M., The Taming of Converse: Reasoning about Two-Way Computations, Proc. Workshop on Logics of Programs, Brooklyn, NY, LNCS no. 193, Springer-Verlag, pp. 413-424, 1985.

[Va87] Vardi, M., Verification of Concurrent Programs: The Automata-theoretic Framework, Proc. IEEE LICS, pp. 167-176, June 87

[VW83] Vardi, M. and Wolper, P., Yet Another Process Logic, in Proc. CMU Workshop on Logics of Programs, Springer LNCS no. 164, pp. 501-512, 1983.

[VW84] Vardi, M. and Wolper, P., Automata Theoretic Techniques for Modal Logics of Programs, STOC 84; journal version in *JCSS*, vol. 32, pp. 183-221, 1986.

[VW86] Vardi, M., and Wolper, P. , An Automata-theoretic Approach to Automatic Program Verification, Proc. IEEE LICS, pp. 332-344, 1986.

[Wo83] Wolper, P., Temporal Logic can be More Expressive, FOCS 81; journal version in Information and Control, vol. 56, nos. 1-2, pp. 72-93, 1983.

[Wo82] Wolper, P., Synthesis of Communicating Processes from Temporal Logic Specifications, Ph.D. Thesis, Stanford Univ., 1982.

[Wo85] Wolper, P., The Tableau Method for Temporal Logic: An Overview, Logique et Analyse, v. 28, June-Sept. 85, pp. 119-136, 1985.

[Wo86] Wolper, P., Expressing Interesting Properties of Programs in Propositional Temporal Logic, ACM Symp. on Princ. of Prog. Lang., pp. 184-193, 1986.

[Wo88] Wolper, P., On the Relation of Programs and Computations to Models of Temporal Logic, manuscript, Univ. of Liege, Belgium, Feb. 88

[Zu86] Zuck, L., Past Temporal Logic, PhD Dissertation, Weizmann Institute, 1986.

Contents

A Hierarchy of Authentication Specifications

Gavin Lowe

Department of Mathematics and Computer Science
University of Leicester, University Road
Leicester, LE1 7RH, UK
E-mail: gavin.lowe@mcs.le.ac.uk

Abstract

Many security protocols have the aim of authenticating one agent to another. Yet there is no clear consensus in the academic literature about precisely what "authentication" means. In this paper we suggest that the appropriate authentication requirement will depend upon the use to which the protocol is put, and identify several possible definitions of "authentication". We formalize each definition using the process algebra CSP, use this formalism to study their relative strengths, and show how the model checker FDR can be used to test whether a system running the protocol meets such a specification.

1 Introduction

Many security protocols have appeared in the academic literature; these protocols often have the aim of achieving *authentication*, i.e., one agent should become sure of the identity of the other. The protocols are designed to succeed even in the presence of a malicious agent, called an *intruder*, who has complete control over the communications network, and so can intercept messages, and introduce new messages into the system, possibly using information from messages he has seen. However, it is rarely made clear precisely what is meant by the term "authentication". This may be dangerous, for a user may assume that the protocol satisfies a stronger condition than the one that was intended by the protocol designer, and so may place more reliance upon the protocol than is justified. In order to alleviate this problem, we study various possible meanings of the word authentication. We formalize these meanings using the process algebra CSP [8].

Suppose an agent A completes a run of an authentication protocol, apparently with B; then what can A deduce about the state of B? Can A deduce that B has recently been alive? Can A deduce that B has recently been running the same protocol as A? Maybe A can deduce that B thought he was running the protocol with A (as opposed to some third party, C). And maybe the two agents agree upon who initiated the exchange, and who responded. Further, they may agree upon the values of some or all of the data items (such as nonces and keys) used in the run. Can A assume that there was a one-one relationship between his runs and B's runs, or might it be the case that A has completed more runs than B? And can A deduce that he and B agreed upon the contents of all messages sent from one to the other?

It is my experience that different researchers will give different answers to the above questions. In order to reason and argue about authentication protocols, we must first of all define what we mean by "authentication". My own view is that an authentication protocol is designed to assure an agent A as to the identity of the other agent B with whom A is running the protocol; therefore, in most cases A should at least be assured that B thought he was running the protocol with A. However, some researchers take the view that it is enough for B to be present, and that A need not receive any further assurance as to B's current state. We should recognize that the different authentication specifications may all be valid goals: there are circumstances in which the weaker specifications are all that is needed; in other circumstances, a stronger specification may be required. However, the designer of any protocol should make it clear which form of authentication is supposed to be achieved.

In this paper we will introduce different terms corresponding to some of the possible meanings of the word "authentication" considered above. We formalize each of these meanings using the process algebra CSP [8]. The use of CSP has two advantages:

- It will allow us to compare the strengths of the different authentication specifications;
- It will open up the possibility of automatically checking whether a system running the protocol achieves the goals stated for it, using a model checker such as FDR [19, 6].

We should stress, though, that most of our results are independent of CSP: CSP just provides a convenient tool for

0-8186-7990-5/97 $10.00 © 1997 IEEE

formalizing and reasoning about the specifications.

Most of the possible meanings of authentication refer to the *recent* state of an agent B when another agent A completes a run of the protocol, apparently with B. For example, a typical authentication specification (which we will call *recent aliveness* below) is that when A completes a run of the protocol, apparently with B, then B has *recently* been running the same protocol. It turns out that the recentness is the hardest part of the specification to formalize in CSP: it normally requires modelling the passage of time. To simplify our presentation, we begin by considering authentication specifications in the case where recentness is not required; for example, we consider a specification (called *aliveness* below) that states that when A completes a run of the protocol, apparently with B, then B has previously (not necessarily recently) been running the same protocol. Later we lift our specifications to include recentness.

It is my experience that most attacks upon protocols break the weaker specifications where recentness is not required; thus from a pragmatic point of view, when using a tool such as FDR to look for attacks upon a protocol, it is sensible to begin by looking for attacks on these weaker specifications, since these tests are faster than in the cases including recentness.

Note that we will not directly consider questions of secrecy in this paper. It is not difficult to formalize secrecy within CSP, using techniques similar to those discussed here. Secrecy and authentication specifications are often broken in the same way: there are a number of attacks upon protocols that lead to an agent A thinking he has established a key with another agent B, when in fact he has been running the protocol with an intruder imitating B, and the intruder ends up knowing the key; this is a failure of authentication, because B has been incorrectly authenticated, and a failure of secrecy, because the intruder has learnt the key that was supposed to remain secret.

In the next section we give precise, although informal, definitions of authentication, first in the case where recentness is not required, and then in the case where recentness is required. In Section 3 we describe the CSP approach to modelling security protocols, and set up some of the mechanism for formalizing the authentication properties. In Section 4 we formalize the authentication properties without recentness, and prove a number of results relating them. Then in Section 5, we lift these specifications to include recentness. We sum up in Section 6. Because of limitations on space, we omit all proofs from this paper; the interested reader is referred to [12].

2 Forms of authentication

In this section we identify four different reasonable meanings of the word "authentication". But first, we need to discuss the setting in which the definitions will apply.

We will consider protocols that aim to authenticate a *responder B* to an *initiator A*, possibly with the help of a third party, a *server*. We use the word "role" to refer to the part an agent is taking in the protocol run (i.e. initiator, responder or server). It should be obvious how to generalize these ideas to include extra agents playing additional roles, or to reverse the direction of authentication.

We do not restrict ourselves to the case where a particular agent may only ever adopt a single role; on the contrary, an agent may act as an initiator in some runs, and as a responder in other runs, and possibly even as a server in others (although this latter case would be unusual).

It is worth drawing a distinction between the free variables appearing in a description of a protocol, and the actual values with which those free variables are instantiated. For example, in a protocol description, the free variable A is often used to represent the initiator; in actual runs, this variable will be instantiated with the identities of actual agents, often different identities in different runs. We will denote free variables representing agents by single letters (A, B, etc.) and will denote actual agents' identities by proper names ($Alice$, Bob, etc.); for other data items, we will use small letters for free variables (na, kab, etc.), and names beginning with a capital letter for actual values (Na, Kab, etc.).

As explained in the introduction, we begin by considering the cases without recentness, and then extend the definitions to include recentness.

2.1 Aliveness

The following is what we consider to be the weakest reasonable definition of authentication.

Definition 2.1 (Aliveness). We say that a protocol guarantees to an initiator A *aliveness* of another agent B if, whenever A (acting as initiator) completes a run of the protocol, apparently with responder B, then B has previously been running the protocol.

Note that B may not necessarily have believed that he was running the protocol with A. Also, B may not have been running the protocol *recently*.

Many protocols fail to achieve even this weak form of authentication. In several cases, this is due to an intruder launching a mirror attack, simply reflecting an agent's messages back at himself; examples appear in [2]. In other, more subtle attacks, an intruder attacks an agent A by using a second run of a protocol with the same agent A, so as to use the second run as an oracle; for example the attack on the BAN version of the Yahalom protocol [3] in [24]. Other attacks are due to more blatant errors; for example, the attack on the SPLICE protocol [25] in [9], which exploits the fact that key delivery messages (from a key server) do not include the identity of the agent whose key is being delivered.

Closely related to the notion of aliveness is the case where, when A completes a run of the protocol, apparently with B, then B has previously been present, but not necessarily running the protocol in question—it may be that B has been running a completely different protocol. This raises the question of interaction between protocols, where an intruder can learn information in one protocol that he can use in an attack on another protocol. In this paper we mainly restrict our attention to systems running a single protocol; we briefly discuss how to extend these techniques to cover systems running several protocols in Section 4.5. However, in general it will be very difficult to prove results about a system running several protocols: we need to be sure that no protocol acts as an oracle for any other.

2.2 Weak agreement

We strengthen the above definition to insist that B agreed he was running the protocol with A.

Definition 2.2 (Weak agreement). We say that a protocol guarantees to an initiator A *weak agreement* with another agent B if, whenever A (acting as initiator) completes a run of the protocol, apparently with responder B, then B has previously been running the protocol, apparently with A.

Note that B may not necessarily have been acting as responder.

Several protocols achieve a guarantee of liveness, but fail to guarantee weak agreement. The normal scenario is that the intruder imitates an agent B to attack A, by using B as an oracle in a parallel run in which the intruder adopts his own identity; thus A believes he has been running the protocol with B, but B does not believe he has been running the protocol with A—B thinks he has been running the protocol with the intruder. Examples include my attack on the Needham-Schroeder Public Key protocol [16] in [10].

2.3 Non-injective agreement

The following definition adds the condition that the two agents agree as to which roles each was taking, and that they agree upon some of the data items used in the exchange.

Definition 2.3 (Non-injective agreement). We say that a protocol guarantees to an initiator A *non-injective agreement* with a responder B on a set of data items ds (where ds is a set of free variables appearing in the protocol description) if, whenever A (acting as initiator) completes a run of the protocol, apparently with responder B, then B has previously been running the protocol, apparently with A, and B was acting as responder in his run, and the two agents agreed on the data values corresponding to all the variables in ds.

Note that this does not guarantee that there is a one-one relationship between the runs of A and the runs of B (hence the adjective "non-injective"): A may believe that he has completed two runs, when B has only been taking part in a single run.

A few protocols achieve a guarantee of weak agreement, but not non-injective agreement. For example, if the original Andrew protocol [21] is adapted to detect mirror attacks, then it achieves weak agreement, but does not achieve non-injective agreement on all the data values: an attack in [3] shows how an intruder can get A to accept a key different from the one used by B.

2.4 Agreement

We use the term, "*injective agreement*", or simply "*agreement*", when we want to insist that there is a one-one relationship between the two agents' runs. This one-one relationship may be important in, for example, financial protocols.

Definition 2.4 (Agreement). We say that a protocol guarantees to an initiator A *agreement* with a responder B on a set of data items ds if, whenever A (acting as initiator) completes a run of the protocol, apparently with responder B, then B has previously been running the protocol, apparently with A, and B was acting as responder in his run, and the two agents agreed on the data values corresponding to all the variables in ds, and each such run of A corresponds to a *unique* run of B.

We will use the term *full agreement* to refer to agreement on all the atomic data items used in the protocol run. For various reasons, we consider this to be the most useful definition of authentication: it insists that the two agents agree on all the essential features of the protocol run, while avoiding specifying features that are hard to achieve and less likely to be required.

A few protocols achieve non-injective agreement, but not (injective) agreement: an agent A is tricked into thinking that B is trying to establish two sessions with him, whereas B was only trying to establish a single run. For example, in the Kerberos protocol [15], the freshness of one agent is guaranteed only by a timestamp; thus, an intruder can replay these messages (within the lifetime of the timestamp) to complete a second run; note that this attack assumes that the agents do not check that the timestamps they receive are distinct from all previous timestamps; Bellovin and Merritt report [1] that early implementations did not perform this check. Similar attacks are described in [13].

2.5 Recentness

Finally, we lift the above definitions to ensure recentness. The meaning of "recent" will depend on the circumstances: sometimes we will take it to mean within the duration of A's run; sometimes we will take it to mean at most t time units before A completed his run, for suitable t, called the *authentication time* (clearly the value of t will be implementation dependent); the designer or implementer of the protocol should make clear what degree of recentness is guaranteed. We use the terms *recent aliveness*, *recent weak agreement*, *recent non-injective agreement*, and *recent agreement* to refer to the above specifications strengthened to insist that B's run was recent, rather than just at some time in the past; we will add the phrase "within t time units" where the protocol guarantees a particular authentication time.

Some protocols meet a particular authentication specification without recentness, but fail to meet the corresponding specification with recentness. For example, consider the following one-step protocol, where kab is a key shared between A and B:

$$\text{Message 1.} \quad A \rightarrow B \; : \; \{A, k\}_{kab} \, .$$

This protocol gives B a guarantee of non-injective agreement on k, but gives no guarantee of recentness, because the message contains no information that B knows to be fresh. The protocol can be strengthened to achieve recent non-injective agreement by adding a timestamp to the message.

Also, in a setting where key compromise is possible, there is a well known attack on the Needham-Schroeder Secret Key Protocol [16], presented in [4]: once a key has been compromised, the intruder may replay messages from an earlier protocol run so as to imitate the initiator; thus (if we make the reasonable assumption that compromising keys takes quite a long time) the protocol guarantees non-injective agreement, but not recent non-injective agreement.

2.6 Discussion

All of the above definitions used a phrase of the form "B has previously been running the protocol". We take this to mean that B has progressed at least as far as the last message that B sends; clearly A can never be assured that B received any subsequent messages. Note that it is possible to define weaker specifications, where A is only assured that B has at least started the protocol.

It should be obvious that the above forms of authentication without recentness are in increasing order of strength, as are the forms of authentication with recentness. Further, each definition using recentness is stronger than the corresponding definition without recentness. We will prove these facts later, after we have formalized the authentication specifications.

2.7 Comparisons

Roscoe [20] introduces the notion of an intensional specification:

> No node can believe a protocol run has completed unless a correct series of messages has occurred (consistent as to all the various parameters) up to and including the last message the given node communicates.

That is, if an agent completes a protocol run, then the protocol must have proceeded essentially as the protocol designer intended: each agent must have seen the expected sequence of messages, all agreeing on the values of atomic data items, and with the correct relative orders of messages.

Roscoe shows how to produce a CSP representation of the above specification; this specification can be checked using the model checker FDR, in a setting very similar to our own.

This is a strong definition of authentication, stronger than our definition of full agreement. It is at least as strong as full agreement, for suppose a protocol satisfies the intensional specification, and that an agent A believes it has completed a protocol run with B. Then from the intensional specification, B's view of the protocol run must agree with A's, and in particular B must have thought he was running the protocol with A, and the two agents agree upon the roles each take, and upon the values of all atomic data items; further, from the way in which intensional specifications are formalized in CSP, there must be a one-one relationship between A's runs and B's runs. Hence full agreement is achieved.

The intensional specification is strictly stronger than full agreement for two reasons:

- In some protocols, an agent A receives an encrypted component that he is not supposed to decrypt, but merely forward to another agent B. In these cases, there is a simple attack where the intruder replaces this component with an arbitrary component, but then reverses the switch when A tries forwarding the component to B. This would not be considered an attack when using the full agreement specification, but is an attack under the intensional specification.
- Consider a protocol where a server sends two consecutive messages to A and to B, respectively. The intensional specification would insist that A and B receive these messages in the same order. However, there is a simple attack where the intruder delays A's message so that it arrives just after

B receives his message. However, this would not be considered an attack when using the full agreement specification.

It is arguable whether the above two attacks should really be considered as attacks. However, there are settings where the precise contents or the precise orderings of messages may be important. We would stress again our philosophy that protocol designers should specify precisely what their protocols are supposed to achieve.

The intensional specification does not, in general, give any guarantee of recentness. For example, consider the one step protocol from Section 2.5; this meets the intensional specification, but does not achieve recent authentication. However, in most protocols recentness is guaranteed by the way in which messages are interleaved: suppose the protocol is such that A sends a message to B and later receives a message back from B (possibly via third parties); suppose further that A is implemented to time out if the run is taking too long, so any actions that occurred since A started the run should be considered recent; then if the intensional specification is met, then it must be the case that the corresponding actions of B occurred after A started this protocol run, so B's actions must indeed have been recent. Roscoe discusses other ways in which intensional specifications can be adapted to deal with time.

Diffie, van Oorschot and Wiener [5] have a similar definition of authentication. They specify that when an agent A completes a run of the protocol, apparently with B, then B has been running the protocol, and the two agents' records of the runs *match*: that is, the messages each sees are the same in all fields relevant to authentication, and the agents agree upon which messages were incoming and which outgoing. However, the definition of matching is such that B may not necessarily think that he was running the protocol with A. Thus this definition is weaker than our weak agreement specification.

Gollmann [7] defines four goals of authentication protocols. We consider here two of these goals, which concern whether a protocol authenticates B to A. Goal G3 states that a cryptographic key associated with B has to be used during the protocol run. (The term "cryptographic key" is intended to be interpreted fairly broadly, so as to include, for example, shared secrets used for authentication; in the case where a key is shared between two agents, the word "associated" is supposed to be interpreted as referring to the agent who actually used the key in question.) The reasoning behind G3 is that B should be authenticated only if an action has occurred that must have been performed by B. Hence this is similar to our recent aliveness specification.

Goal G4 states that the origin of all messages in the protocol has to be authenticated. In other words, if A receives a message, apparently from B, then B previously tried to send this message to A. It is left vague whether B necessarily sent this message recently, and whether A may receive two messages for a single message sent by B, and so the above goal can be interpreted in different ways. However, this goal is clearly similar to Roscoe's intensional specification.

Paulson [17, 18] uses the theorem prover Isabelle to analyse security protocols. He proves properties of the form: if A receives a message of a certain form, which appears to come from B, then B indeed sent that message. Thus his notion of authentication is similar to our non-injective agreement. (The form of authentication he considers is necessarily non-injective, because of a feature of his model that allows an agent to respond several times to a single message; there is no real reason why this feature could not be changed, in which case he would be able to deal with injective agreement, although possibly at additional computational expense.)

3 Modelling protocols using CSP

In this section we briefly review the method we use for modelling security protocols using CSP. For a fuller description, the reader is referred to [11]. All the authentication specifications we are considering are safety specifications (as opposed to liveness specifications); we will therefore be working in the traces model of CSP, which is adequate for expressing safety properties.

As an example, we consider the 3 message version of the Needham-Schroeder Public Key protocol [16, 10]. The protocol can be defined by:

$$\begin{array}{lll} \text{Message 1.} & A \rightarrow B & : A, B, \{na, A\}_{PK(B)} \\ \text{Message 2.} & B \rightarrow A & : B, A, \{na, nb\}_{PK(A)} \\ \text{Message 3.} & A \rightarrow B & : A, B, \{nb\}_{PK(B)} \,. \end{array}$$

Each agent is represented by a CSP process that sends and receives the appropriate protocol messages, augmented with extra events that signal the beliefs of the agent. For example:

$$\begin{array}{l} INITIATOR_0(A, na, pka) \mathrel{\widehat=} \\ \quad \square_{B:Agent}\, env.A.(Env0, B) \rightarrow \\ \quad comm.A.B.(Msg1, Encrypt.(PK(B), \langle na, A\rangle)) \rightarrow \\ \quad \square_{nb:Nonce} \\ \qquad comm.B.A.(Msg2, Encrypt.(pka, \langle na, nb\rangle)) \rightarrow \\ \quad signal.Running.INIT\text{-}role.A.B.na.nb \rightarrow \\ \quad comm.A.B.(Msg3, Encrypt.(PK(B), \langle nb\rangle)) \rightarrow \\ \quad signal.Commit.INIT\text{-}role.A.B.na.nb \rightarrow \\ \quad SKIP \end{array}$$

The way in which protocol messages are represented by CSP events should be obvious; for example, the event

$$comm.A.B.(Msg1, Encrypt.(PK(B), \langle na, A\rangle))$$

represents the message:

$$\text{Message 1.} \quad A \to B \;:\; \{na, A\}_{PK(B)} \;.$$

The channel *signal* is used to express properties of the beliefs of the agent. An event of the form *signal.Running.role.A.B.na.nb* represents that the agent A believes that he is taking the role *role* in a run of the protocol with B, using nonces *na* and *nb*; such an event is performed immediately before the last message that A sends. An event of the form *signal.Commit.role.A.B.na.nb* represents that the agent A thinks he has successfully completed a run of the protocol, taking role *role*, with B, using the nonces *na* and *nb*; such an event is performed after A's last event in the protocol. These *signal* events will be used in formalizing our authentication specifications. The commit and running signals include (in some standard order) all the data items that the agents are supposed to agree on.

In order to represent the fact that the intruder may interfere with the normal operation of the protocol, a CSP-renaming is applied to the above processes, with the effect that outputs may be intercepted by the intruder (and so fail to reach their intended destination), and inputs may be faked (that is, produced by the intruder).

The parameters of these processes are instantiated with appropriate actual values; this allows us, for example, to define several *INITIATOR* processes with the identity *Alice*, representing that the agent *Alice* may run the protocol several times.

The processes are then combined together in parallel with a process representing the most general intruder who can interact with the protocol. This intruder may: oversee messages passing in the system, and so learn new messages; intercept messages; decrypt messages he has seen, if he has the appropriate key, and so learn new (sub-)messages; encrypt messages he has seen with keys that he knows so as to create new messages; and use the knowledge he has accumulated to send fake messages.

FDR can be used to check whether the resulting system correctly achieves the goals of the protocol. Formalizing these goals is the subject of the rest of this paper.

4 Formalizing authentication specifications without recentness

As explained in the introduction, we begin by formalizing authentication specifications that ignore issues of recentness. In the next section we adapt these specifications to include recentness.

We introduce two pieces of notation that we will need. If P is a process, we define αP to be its alphabet. We write P^n for the interleaving of n copies of P:

$$P^n \;\hat{=}\; \underbrace{P \,|||\, \ldots \,|||\, P}_{n}$$

Formally: $P^0 \;\hat{=}\; STOP$ and $P^{n+1} \;\hat{=}\; P \,|||\, P^n$. We will take $\alpha P^0 = \alpha P$.

We now formalize the different authentication specifications; we consider the specifications in the reverse order to that in the introduction. We will consider a pair of agents, A and B, taking the roles *A-role* and *B-role*, respectively, and consider the question of whether the protocol correctly authenticates B to A.

4.1 Agreement

We begin with the *agreement* specification. Suppose the protocol uses two data items d_1 and d_2, and we want to test whether the protocol guarantees to an arbitrary agent A taking the role *A-role*, agreement with an agent B, taking the role *B-role*, on both the data items d_1 and d_2. This will be the case if, whenever A completes a run of the protocol apparently with B, then B has previously been running the protocol, apparently with A, using the same values for the data items, and there is a one-one relationship between A's runs and B's runs. Now, B running the protocol, taking the role *B-role*, apparently with A, using particular values d_1' and d_2' for the data items d_1 and d_2, is represented by the event *signal.Running.B-role.B.A.*$d_1'.d_2'$; similarly, A completing the protocol, taking the role *A-role*, apparently with B, using the same values for the data items, is represented by the event *signal.Commit.A-role.A.B.*$d_1'.d_2'$. Thus we will want to ensure that the abstraction of the system to the appropriate alphabet satisfies the following specification:

$$\begin{aligned}&Agreement(B\text{-}role, A\text{-}role, \{d_1, d_2\})(B) \;\hat{=}\\ &\quad signal.Running.B\text{-}role.B?A?d_1'?d_2' \to\\ &\quad signal.Commit.A\text{-}role.A.B.d_1'.d_2' \to STOP\;.\end{aligned}$$

This specification says that whenever A performs an event of the form *signal.Commit.A-role.A.B.*$d_1'.d_2'$, then B has previously performed a corresponding event *signal.Running.B-role.B.A.*$d_1'.d_2'$. That is, whenever A signals that he thinks he has completed the protocol, taking the role *A-role*, with B, using the data values d_1' and d_2', then B has previously been running the protocol, taking the role *B-role*, with A, using the same data values. (Remember that we are working in the traces model of CSP, so the above specification does not insist that a *signal.Commit* event must occur.)

Figure 1 illustrates the meanings of the arguments.

Consider the case where each agent may take part in a single run of the protocol. We may test whether the protocol correctly authenticates a particular agent B, taking role *B-role*, to an arbitrary agent A, taking role *A-role*,

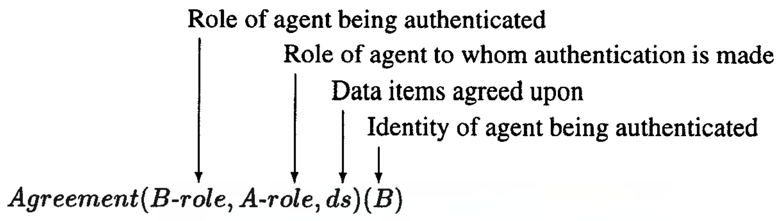

Role of agent being authenticated
Role of agent to whom authentication is made
Data items agreed upon
Identity of agent being authenticated
Agreement(B-role, A-role, ds)(B)

Figure 1. The arguments of *Agreement*

with agreement on all the data items, by testing:

$$\begin{array}{l} Agreement(B\text{-}role, A\text{-}role, \{d_1, d_2\})(B) \sqsubseteq \\ \quad SYSTEM \setminus (\Sigma - X) \\ \text{where} \\ X = \alpha Agreement(B\text{-}role, A\text{-}role, \{d_1, d_2\})(B)), \end{array}$$

where Σ is the set of all events. (The fact that A's identity is not included in the parameters of the process *Agreement* has the effect of making the above refinement check test whether B is correctly authenticated to *all* agents who take the role *A-role*. The reason why we test whether a *single* agent B is correctly authenticated, rather than testing whether all agents who take the role *B-role* are correctly authenticated, is purely pragmatic: the latter check would become very slow for large systems. The reason for the slightly odd parameterization is to keep consistency with the tool Casper, which we will discuss in the concluding section, which automatically produces CSP descriptions of security protocols, and specifications similar to the above, from a more abstract definition.)

The above form of refinement check will be very common in the rest of this paper, so we introduce a shorthand for it.

$$\begin{array}{l} SYSTEM \text{ meets } SPEC \mathrel{\widehat=} \\ \quad SPEC \sqsubseteq SYSTEM \setminus (\Sigma - \alpha SPEC). \end{array}$$

So the above test is written:

$$\begin{array}{l} SYSTEM \text{ meets} \\ \quad Agreement(B\text{-}role, A\text{-}role, \{d_1, d_2\})(B). \end{array}$$

For example, to test whether the Needham-Schroeder Public Key Protocol guarantees to an arbitrary responder B agreement with the initiator *Alice* on the nonces *na* and *nb*, we should test:

$$\begin{array}{l} SYSTEM \text{ meets} \\ \quad Agreement \\ \qquad (INIT\text{-}role, RESP\text{-}role, \{na, nb\})(Alice). \end{array}$$

It turns out that the above specification is not met (the protocol does not correctly authenticate the initiator), as can be seen from the attack in [10, 11].

We now generalize the above refinement checks to consider the case where the agent being authenticated can perform more than one run of the protocol. In this case, we want to ensure that *every Commit* event is matched by a corresponding *Running* event. If B can perform at most n runs, then there will be at most n *Running* events, so we want to consider the following refinement check:

$$\begin{array}{l} SYSTEM \text{ meets} \\ \quad Agreement(B\text{-}role, A\text{-}role, \{d_1, d_2\})(B)^n. \end{array}$$

Note that this specification guarantees that only one *Commit* event may correspond to each *Running* event, thus ensuring that the initiator commits to only a single session for each run of the protocol performed by the responder.

In some cases it will be the case that we want to ensure that the agents involved agree on the values of some data items, but allow them to disagree on other data items. For example, if we want the agents to agree on d_1, but allow them to disagree on d_2, then we may use the specification:

$$\begin{array}{l} Agreement(B\text{-}role, A\text{-}role, \{d_1\})(B) \mathrel{\widehat=} \\ \quad signal.Running.B\text{-}role.B?A?d_1'?d_2'' \to \\ \quad signal.Commit.A\text{-}role.A.B.d_1'?d_2' \to STOP \end{array}$$

If we do not insist that they agree on any data items, then we may use the specification:

$$\begin{array}{l} Agreement(B\text{-}role, A\text{-}role, \{\})(B) \mathrel{\widehat=} \\ \quad signal.Running.B\text{-}role.B?A?d_1''?d_2'' \to \\ \quad signal.Commit.A\text{-}role.A.B?d_1'?d_2' \to STOP \end{array}$$

In general, the third argument of the *Agreement* macro should be a subset of the data variables appearing in the *signal.Running* and *signal.Commit* events.

In the above, we have considered a protocol using a pair of data values; it should be obvious how to generalize to an arbitrary set of data variables *ds*. We will write *signal.Running.B-role.B.A.ds'* to indicate agent B thinking that he is running the protocol using values *ds'* for the data variables *ds*, and similarly for the *Commit* signal (strictly speaking, *ds* is a *set*, and *ds'* is an enumeration of the elements of *ds*, in some standard order).

It should be obvious that if a protocol guarantees agreement on some set ds_1 of data values, then it will guarantee agreement on any smaller set ds_2. The following lemma formalizes this.

Lemma 4.1. If $ds_2 \subseteq ds_1$ and

$$\begin{aligned}&SYSTEM \text{ meets}\\ &\quad Agreement(B\text{-}role, A\text{-}role, ds_1)(B)^n,\end{aligned}$$

then

$$\begin{aligned}&SYSTEM \text{ meets}\\ &\quad Agreement(B\text{-}role, A\text{-}role, ds_2)(B)^n.\end{aligned}$$

4.2 Non-injective agreement

In the previous subsection, we insisted that there was a one-one relationship between the runs of A and those of B. In some settings, this injectivity may not be important, and so we will allow A to commit an arbitrary number of times, for each run of B:

$$\begin{aligned}&NonInjectiveAgreement\\ &\qquad (B\text{-}role, A\text{-}role, \{d_1, d_2\})(B) \mathrel{\widehat{=}}\\ &\quad signal.Running.B\text{-}role.B?A?d_1'?d_2' \to\\ &\quad RUN(\{signal.Commit.A\text{-}role.A.B.d_1'.d_2'\}).\end{aligned}$$

That is, A may think he has completed an arbitrary number of runs, taking role A-*role*, with B, using data values d_1' and d_2' (i.e., perform an arbitrary number of $signal.Commit.A\text{-}role.A.B.d_1'.d_2'$ events) if B thinks he has been running the protocol, taking role B-*role*, with A, using the same data values, at least once (i.e., if he has performed at least one $signal.Running.B\text{-}role.B.A.d_1'.d_2'$ event).

This can be adapted to insist upon agreement of only some of the data values; for example:

$$\begin{aligned}&NonInjectiveAgreement(B\text{-}role, A\text{-}role, \{d_1\})(B) \mathrel{\widehat{=}}\\ &\quad signal.Running.B\text{-}role.B?A?d_1'?d_2'' \to\\ &\quad RUN(\{signal.Commit.A\text{-}role.A.B.d_1'.d_2' \mid\\ &\qquad\qquad d_2' \in Data\}).\end{aligned}$$

Non-injective agreement can then be tested analogously to agreement:

$$\begin{aligned}&SYSTEM \text{ meets}\\ &\quad NonInjectiveAgreement(B\text{-}role, A\text{-}role, ds)(B)^n.\end{aligned}$$

As with injective agreement, agreement on some set of data items implies agreement on any smaller set.

Lemma 4.2. If $ds_2 \subseteq ds_1$ and:

$$\begin{aligned}&SYSTEM \text{ meets}\\ &\quad NonInjectiveAgreement\\ &\qquad (B\text{-}role, A\text{-}role, ds_1)(B)^n,\end{aligned}$$

then:

$$\begin{aligned}&SYSTEM \text{ meets}\\ &\quad NonInjectiveAgreement\\ &\qquad (B\text{-}role, A\text{-}role, ds_2)(B)^n.\end{aligned}$$

Non-injective agreement is a weaker requirement that agreement; this is formalized by the following lemma.

Lemma 4.3. If

$$SYSTEM \text{ meets } Agreement(B\text{-}role, A\text{-}role, ds)(B)^n,$$

then

$$\begin{aligned}&SYSTEM \text{ meets}\\ &\quad NonInjectiveAgreement(B\text{-}role, A\text{-}role, ds)(B)^n.\end{aligned}$$

4.3 Weak agreement

We can weaken the previous specification, so that A receives no guarantee as to which role B thought he was taking, and no guarantee regarding agreement on data. We define the specification:

$$\begin{aligned}&WeakAgreement(A\text{-}role)(B) \mathrel{\widehat{=}}\\ &\quad signal.Running?B\text{-}role!B?A?ds' \to\\ &\quad RUN(\{signal.Commit.A\text{-}role.A.B.ds \mid\\ &\qquad\qquad ds \in Data^*\}).\end{aligned}$$

That is, A may think he has completed an arbitrary number of runs, taking role A-*role*, with B, if B thinks he has been running the protocol, taking some role B-*role*, with A, possibly using different data values, at least once.

Weak agreement can then be checked using the refinement test:

$$SYSTEM \text{ meets } WeakAgreement(A\text{-}role)(B)^n.$$

The following lemma shows that weak agreement is indeed a weakening of non-injective agreement.

Lemma 4.4. Suppose the agent B can perform a maximum of m runs with role B-*role*, and n runs with other roles. If:

$$\begin{aligned}&SYSTEM \text{ meets}\\ &\quad NonInjectiveAgreement(B\text{-}role, A\text{-}role, ds)(B)^m,\end{aligned}$$

then:

$$SYSTEM \text{ meets } WeakAgreement(A\text{-}role)(B)^{m+n}.$$

4.4 Aliveness

Finally, we weaken the specification still further, so that A receives no guarantee that B thought he was running the protocol with A; A merely receives a guarantee that B was previously running the protocol with somebody:

$$\begin{aligned}&Aliveness(A\text{-}role)(B) \mathrel{\widehat{=}}\\ &\quad signal.Running?B\text{-}role!B?C?ds' \to\\ &\quad RUN(\{signal.Commit.A\text{-}role.A.B.ds \mid\\ &\qquad\qquad ds \in Data^* \land A \in Agent\})\end{aligned}$$

That is, A may think he has completed an arbitrary number of runs, taking role $A\text{-}role$, with B, if B thinks he has previously been running the protocol.

Agreement can then be checked using the refinement test:

$$SYSTEM \text{ meets } Agreement(A\text{-}role)(B)^n.$$

The following lemma shows that weak agreement is indeed a strengthening of aliveness.

Lemma 4.5. If

$$SYSTEM \text{ meets } WeakAgreement(A\text{-}role)(B)^n,$$

then

$$SYSTEM \text{ meets } Aliveness(A\text{-}role)(B)^n.$$

4.5 Disagreeing over the protocol being run

All of the above discussion has assumed that we are operating in a system with only a single protocol. However, the above techniques are easily extended to cover the more general case.

We may augment the *signal* events with an extra field representing the identity of the protocol that an agent thinks he's running, and can then adapt the authentication specifications appropriately. For example, the following specification is an adaptation of the *Aliveness* specification, which guarantees that when one agent completes a run of protocol *protId*, then the other agent was running the same protocol:

$$\begin{aligned} &AlivenessSameProtocol(protId, A\text{-}role)(B) \mathrel{\widehat=} \\ &\quad signal.Running.protId?B\text{-}role!B?C?ds' \rightarrow \\ &\quad RUN(\{signal.Commit.protId.A\text{-}role.A.B.ds \mid \\ &\qquad\qquad ds \in Data^* \wedge A \in Agent\}) \end{aligned}$$

Similarly, the following specification allows the agents to be running different protocols:

$$\begin{aligned} &AlivenessSameProtocol(protId, A\text{-}role)(B) \mathrel{\widehat=} \\ &\quad signal.Running?protId'?B\text{-}role!B?C?ds' \rightarrow \\ &\quad RUN(\{signal.Commit.protId.A\text{-}role.A.B.ds \mid \\ &\qquad\qquad ds \in Data^* \wedge A \in Agent\}). \end{aligned}$$

5 Formalizing authentication specifications with recentness

As described in the introduction, the specifications from the previous section do not insist that the runs are in any way contemporaneous: one agent may commit even though the other has not performed any events *recently*. In this section we strengthen the specifications to insist that the runs are contemporaneous. There are a number of ways of achieving this: we outline two possible approaches in the following two subsections; two other possible approaches are described in [12].

Throughout this section we will make an implementation assumption that no run lasts for too long: if a run lasts for longer than some allowable maximum, then the agent(s) in question should time out and abort the run. This will mean that any two events within the duration of a particular run should be considered to be recent to one another.

5.1 Recentness through freshness

In some cases, the recentness of an agent's run can be guaranteed by agreement on a fresh data value. For example, consider the example of the Needham-Schroeder Public Key Protocol, and suppose we have shown that:

$$\begin{aligned} &SYSTEM \text{ meets} \\ &\quad Agreement \\ &\qquad (RESP\text{-}role, INIT\text{-}role, \{na, nb\})(Bob)^n, \end{aligned}$$

for some agent *Bob*. That is: if any initiator A completes a run of the protocol, apparently with *Bob*, using particular values for the nonces, then A can be sure that at some time in the past, *Bob* believed that he was acting as responder in a run of the protocol with A, using the same values for the nonces. However, it is assumed that A invented the value of na as a fresh value for this particular run. Hence *Bob*'s run must have occurred at some time after A invented this nonce. Thus, from the implementation assumption about A's runs not lasting too long, *Bob* must have performed this run *recently*.

This style of argument can be used with both the *Agreement* and *NonInjectiveAgreement* specifications, and will guarantee recentness whenever the agents agree on some data value that is freshly invented by the agent to whom the authentication is made. However, when there is no agreement upon a fresh variable, an alternative approach has to be found.

5.2 Timed authentication

We now consider an alternative method of ensuring recentness, namely through introducing a representation of time into the CSP model of the system. We represent the passage of one unit of time by an event *tock*. We then interpret "recently" to mean within the last *AuthTime* time units, where *AuthTime* is a parameter provided by the protocol tester, called the *authentication time*. Below, we will check whether the protocol correctly achieves recent authentication by performing a test of the form: whenever A

commits to a session, apparently with B, then B was running the protocol within the last $AuthTime$ *tock*s (possibly with other conditions, corresponding to the different forms of authentication).

In order for the protocol to meet such a specification, we will assume that each agent will timeout if a particular run lasts for longer than some time $MaxRunTime$. (Clearly if such an implementation assumption is not met, we have little chance of meeting a timed specification.) Below, we will define a function that takes an untimed definition of an agent, and gives a timed version, with such a timeout.

First, we define two subsidiary processes: $TOCKS(n)$ will perform at most n *tock*s before terminating; $TSKIP$ will perform an arbitrary number of *tock*s before terminating:

$$\begin{array}{l} TOCKS(n) \mathrel{\widehat{=}} \\ \quad \text{if } n = 0 \text{ then } SKIP \\ \quad \text{else } (tock \to TOCKS(n-1) \mathrel{\Box} SKIP), \\ TSKIP \mathrel{\widehat{=}} tock \to TSKIP \mathrel{\Box} SKIP. \end{array}$$

We now define a function $AddTime$ that turns an untimed representation P of an agent into a timed one. The timed process initially allows an arbitrary amount of time to pass before a run begins. Once the run has begun, at most $MaxRunTime$ *tock*s should occur during the run itself: if an extra *tock* occurs, then the agent should timeout, and just allow time to pass before terminating. This may be defined as follows:[1]

$$\begin{array}{l} AddTime(P, MaxRunTime) \mathrel{\widehat{=}} \\ \quad tock \to AddTime(P, MaxRunTime) \\ \quad \Box \\ \quad ((P \mathbin{|||} TOCKS(MaxRunTime)) \\ \qquad \triangle\; tock \to TSKIP). \end{array}$$

(In fact, the above definition allows the agent to nondeterministically abort the run after fewer than $MaxRunTime$ time units; this corresponds to the agent aborting for some reason we are not modelling, such as user intervention. However, if more than $MaxRunTime$ *tock*s occur, then one of the *tock*s must trigger the timeout. Note, though, that the trace set of the process is not affected by this possibility of early timeout.)

We apply the $AddTime$ function to all the processes representing untimed agents, so as to build timed versions; we then combine these timed agents together to build a timed system.

We are now ready to produce the timed authentication specifications. We use the $AddTime$ function to lift the untimed agreement specifications to timed versions. For example:

$$\begin{array}{l} TimedAgreement(B\text{-}role, A\text{-}role, ds, AuthTime)(B) \\ \widehat{=} \\ AddTime(Agreement(B\text{-}role, A\text{-}role, ds)(B), \\ \qquad AuthTime). \end{array}$$

The above process allows at most $AuthTime$ *tock*s during the execution of $Agreement(B\text{-}role, A\text{-}role, ds)(B)$, i.e., between the $signal.Running$ and $signal.Commit$ events; thus it specifies that the $Commit$ signal must occur within $AuthTime$ *tock*s of the $Running$ signal.

We can test whether a system correctly achieves timed agreement by testing whether the appropriate abstraction of the system (built from timed agents) refines n copies of the above specification:

$$\begin{array}{l} SYSTEM \text{ meets} \\ \quad TimedAgreement \\ \qquad (B\text{-}role, A\text{-}role, ds, AuthTime)(B)^n. \end{array}$$

But now we must redefine the $(\_)^n$ notation so as to synchronize all the individual specifications on *tock* events:

$$P^n \mathrel{\widehat{=}} \underbrace{P \underset{\{tock\}}{\|} \cdots \underset{\{tock\}}{\|} P}_{n}.$$

Formally: $P^0 \mathrel{\widehat{=}} RUN(\{tock\})$ and $P^{n+1} \mathrel{\widehat{=}} P \underset{\{tock\}}{\|} P^n$.

The other untimed authentication specifications can similarly be lifted to timed specifications by using the *AddTime* function:

$$\begin{array}{l} TimedNonInjectiveAgreement \\ \quad (B\text{-}role, A\text{-}role, ds, AuthTime)(B) \\ \widehat{=} \\ AddTime(\\ \quad NonInjectiveAgreement(B\text{-}role, A\text{-}role, ds)(B), \\ \quad AuthTime), \\ TimedWeakAgreement(A\text{-}role, AuthTime)(B) \\ \widehat{=} \\ AddTime(WeakAgreement(A\text{-}role)(B), AuthTime), \\ TimedAliveness(A\text{-}role, AuthTime)(B) \\ \widehat{=} \\ AddTime(Aliveness(A\text{-}role)(B), AuthTime). \end{array}$$

The following lemma is analogous to the lemmas of Section 4.

Lemma 5.1.

- If $ds' \subseteq ds$ and

$$\begin{array}{l} SYSTEM \text{ meets} \\ \quad TimedAgreement \\ \qquad (B\text{-}role, A\text{-}role, ds, AuthTime)(B)^n \end{array}$$

[1]The process $Q \triangle R$ represents an interrupt mechanism; it initially acts like Q, but may be interrupted by the performance of any event of R, and will then continue to act like R.

then

$$SYSTEM \text{ meets } TimedAgreement(B\text{-}role, A\text{-}role, ds', AuthTime)(B)^n.$$

- If $ds' \subseteq ds$ and

$$SYSTEM \text{ meets } TimedNonInjectiveAgreement(B\text{-}role, A\text{-}role, ds, AuthTime)(B)^n$$

then

$$SYSTEM \text{ meets } TimedNonInjectiveAgreement(B\text{-}role, A\text{-}role, ds', AuthTime)(B)^n$$

- If

$$SYSTEM \text{ meets } TimedAgreement(B\text{-}role, A\text{-}role, ds, AuthTime)(B)^n$$

then

$$SYSTEM \text{ meets } TimedNonInjectiveAgreement(B\text{-}role, A\text{-}role, ds, AuthTime)(B)^n$$

- Suppose the agent B can perform a maximum of m runs with role B-*role*, and n runs with other roles. If

$$SYSTEM \text{ meets } TimedNonInjectiveAgreement(B\text{-}role, A\text{-}role, ds, AuthTime)(B)^m$$

then

$$SYSTEM \text{ meets } TimedWeakAgreement(A\text{-}role, AuthTime)(B)^{m+n}$$

- If

$$SYSTEM \text{ meets } TimedWeakAgreement(A\text{-}role, AuthTime)(B)^n$$

then

$$SYSTEM \text{ meets } TimedAliveness(A\text{-}role, AuthTime)(B)^n.$$

The timed authentication specifications are antimonotonic in the time parameter.

Lemma 5.2. Let $TimedSpec(\_)$ be one of the timed authentication specifications, with parameter representing the time (for example, $TimedSpec(t) = TimedAliveness(A\text{-}role, t)(B)$), and let $t \leq t'$. Then if

$$SYSTEM \text{ meets } TimedSpec(t)^n$$

then

$$SYSTEM \text{ meets } TimedSpec(t')^n.$$

Finally, the timed specifications imply the corresponding untimed specifications.

Lemma 5.3. Let *Spec* be one of the untimed authentication specifications, and let *TimedSpec* be the corresponding timed specification, i.e., $TimedSpec = AddTime(Spec, AuthTime)$. If $SYSTEM$ meets *TimedSpec* then $SYSTEM$ meets *Spec*.

6 Conclusions

In this paper we have considered the question of the meaning of the term "entity authentication". We argued that different authentication requirements may be appropriate for different circumstances, and identified a number of possible authentication specifications.

We formalized each authentication specification using the process algebra CSP, used this formalism to study their relative strengths, and showed how to use the model checker FDR to test whether a system running a protocol achieves the various authentication requirements.

Our authentication specifications have much in common with Schneider's work [22, 23]. He uses a trace specification T authenticates R to mean that events from T authenticate events from R: events from T can happen only if there have been preceding events from R:

$$T \text{ authenticates } R \mathrel{\widehat{=}} tr \upharpoonright R = \langle\rangle \Rightarrow tr \upharpoonright T = \langle\rangle.$$

Note that the above test does not capture our notion of injectivity: if an event from R occurs, then there may be several subsequent events from T. Nor does it capture our notion of recentness: the events from T may occur some time after the corresponding events from R. However, it is not hard to adapt the specification to as to capture both injectivity and recentness.

The *NonInjectiveAgreement*, *WeakAgreement* and *Aliveness* tests of this paper may be seen as testing whether a *Commit* signal authenticates a corresponding

Running signal. This is made formal as follows:

$$
\begin{array}{l}
SYSTEM \text{ meets} \\
\quad NonInjectiveAgreement(B\text{-}role, A\text{-}role, ds)(B) \\
iff \\
\forall A \in Agent\,;\, ds' \in Data^* \bullet \\
\quad SYSTEM \text{ sat } \{signal.Commit.A\text{-}role.A.B.ds'\} \\
\qquad\qquad\qquad \text{authenticates} \\
\qquad\qquad\qquad \{signal.Running.B\text{-}role.B.A.ds'\},
\end{array}
$$

$$
\begin{array}{l}
SYSTEM \text{ meets } WeakAgreement(A\text{-}role)(B) \\
iff \\
\forall A \in Agent \bullet \\
\quad SYSTEM \text{ sat} \\
\qquad \{signal.Commit.A\text{-}role.A.B.ds \mid ds \in Data^*\} \\
\qquad \text{authenticates} \\
\qquad \{signal.Running.B\text{-}role.B.A.ds' \mid \\
\qquad\quad B\text{-}role \in ROLE \wedge ds' \in Data^*\},
\end{array}
$$

$$
\begin{array}{l}
SYSTEM \text{ meets } Aliveness(A\text{-}role)(B) \\
iff \\
SYSTEM \text{ sat} \\
\quad \{signal.Commit.A\text{-}role.A.B.ds \mid \\
\qquad ds \in Data^* \wedge A \in Agent\} \\
\quad \text{authenticates} \\
\quad \{signal.Running.B\text{-}role.B.C.ds' \mid \\
\qquad B\text{-}role \in ROLE \wedge \\
\qquad ds' \in Data^* \wedge C \in Agent\}.
\end{array}
$$

Casper is a program that takes an abstract description of a security protocol, and produces a corresponding CSP description, suitable for checking using FDR. Each of the authentication specifications considered in this paper have been implemented within Casper. Thus Casper and FDR can be used together to test which, if any, authentication specifications are met by a particular protocol. A number of case studies using these techniques are available via the Casper World Wide Web page [14]; these case studies demonstrate the practicality of our methods.

When analyzing protocols using FDR, we make a number of implementation assumptions, which are reflected in the way we model the intruder and the honest agents. For example: above we made the assumption that honest agents would time out if runs are taking too long; we normally assume that the intruder is unable to decrypt messages unless he has the appropriate key; and we normally assume that the encryption method used has no "interesting" algebraic properties. However, none of these assumptions are really necessary: by altering the way we model the system, we can remove these assumptions. Note, though, that the authentication specifications discussed in this paper are independent of the implementation assumptions, and so would still be applicable under different assumptions.

Acknowledgements

I would like to thank Steve Schneider, Bill Roscoe and Dieter Gollmann for interesting discussions and useful comments on the work in this paper.

References

[1] S. M. Bellovin and M. Merritt. Limitations of the Kerberos authentication system. *ACM Computer Communications Review*, 20(5):119–132, 1990.

[2] R. Bird, I. Gopal, A. Herzberg, P. A. Janson, S. Kutten, R. Mulva, and M. Yung. Systematic design of a family of attack-resistant authentication protocols. *IEEE Journal on Selected Areas in Communications*, 11(5):679–693, 1993.

[3] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *Proceedings of the Royal Society of London A*, 426:233–271, 1989. A preliminary version appeared as Digital Equipment Corporation Systems Research Center report No. 39, 1989.

[4] D. E. Denning and G. M. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536, 1981.

[5] W. Diffie, P. C. van Oorschot, and M. J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, 2:107–125, 1992.

[6] Formal Systems (Europe) Ltd. *Failures-Divergence Refinement—FDR 2—User Manual*, 1997. Available via URL `ftp://ftp.comlab.ox.ac.uk/pub/Packages/FDR`.

[7] D. Gollmann. What do we mean by entity authentication? In *IEEE Symposium on Research in Security and Privacy*, 1996.

[8] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[9] T. Hwang and Y.-H. Chen. On the security of SPLICE/AS—the authentication system in WIDE Internet. *Information Processing Letters*, 53:97–101, 1995.

[10] G. Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters*, 56:131–133, 1995.

[11] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proceedings of TACAS*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer Verlag, 1996. Also in *Software—Concepts and Tools*, 17:93–102, 1996.

[12] G. Lowe. A hierarchy of authentication specifications. Technical Report 1996/33, Department of Mathematics and Computer Science, University of Leicester, 1996.

[13] G. Lowe. A family of attacks upon authentication protocols. Technical Report 1997/5, Department of Mathematics and Computer Science, University of Leicester, 1997.

[14] G. Lowe. Casper: A compiler for the analysis of security protocols, 1997. In this volume. World Wide Web home page at URL `http://www.mcs.le.ac.uk/~glowe/Security/Casper/index.html`.

[15] S. P. Miller, C. Neumann, J. I. Schiller, and J. H. Saltzer. Kerberos authentication and authorization system. Project Athena Technical Plan Section E.2.1, MIT, 1987. Available from URL `ftp://athena-dist.mit.edu/pub/kerberos/doc/techplan.PS`.

[16] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[17] L. Paulson. Proving Properties of Security Protocols by Induction. Technical Report 409, University of Cambridge Computer Laboratory, 1996.

[18] L. Paulson. Mechanized Proofs of Security Protocols: Needham-Schroeder with Public Keys. Technical Report 413, University of Cambridge Computer Laboratory, 1997.

[19] A. W. Roscoe. Model-checking CSP. In *A Classical Mind, Essays in Honour of C. A. R. Hoare*. Prentice-Hall, 1994.

[20] A. W. Roscoe. Intensional specification of security protocols. In *9th IEEE Computer Security Foundations Workshop*, pages 28–38, 1996.

[21] M. Satyanarayanan. Integrating security in a large distributed system. *ACM Transactions on Computer Systems*, 7(3):247–280, 1989.

[22] S. Schneider. Security properties and CSP. In *IEEE Computer Society Symposium on Security and Privacy*, Oakland, 1996.

[23] S. Schneider. Using CSP for protocol analysis: the Needham-Schroeder Public Key Protocol. Technical Report CSD-TR-96-14, Royal Holloway, 1996.

[24] P. Syverson. A taxonomy of replay attacks. In *Proceedings of the 7th IEEE Computer Security Foundations Workshop*, pages 131–136, 1994.

[25] S. Yamaguchi, K. Okayama, and H. Miyahara. Design and implementation of an authentication system in WIDE Internet environment. In *Proc. 10th IEEE Region Conf. on Computer and Communication Systems*, 1990.

Using Temporal Logics of Knowledge in the Formal Verification of Security Protocols

Clare Dixon, Mari-Carmen Fernández Gago, Michael Fisher and Wiebe van der Hoek
Department of Computer Science
The University of Liverpool, Liverpool L69 7ZF, UK
{clare,mcarmen,michael,wiebe}@csc.liv.ac.uk

Abstract

Temporal logics of knowledge are useful for reasoning about situations where the knowledge of an agent or component is important, and where change in this knowledge may occur over time. Here we use temporal logics of knowledge to reason about security protocols. We show how to specify part of the Needham-Schroeder protocol using temporal logics of knowledge and prove various properties using a clausal resolution calculus for this logic.

1. Introduction

Improved communication infrastructures encourage parties to interchange more and more sensitive data, such as payment instructions in e-commerce, strategic information between commercial partners, or personal information in, for instance, medical applications. Issues such as authentication of the partners in a protocol, together with the confidentiality of information, therefore become increasingly important. Consequently, cryptographic protocols are commonly used to distribute keys and authenticate agents and data over hostile networks. Although the protocols used often appear watertight, many examples are known of sensitive applications that were 'cracked' and had to be furnished with new, 'improved', protocols. It is obvious that in such information-sensitive applications as above, one prefers to *formally prove* that certain information can not be eavesdropped by unwanted third parties.

The application of logical tools to the analysis of security protocols was pioneered by Burrows, Abadi and Needham. In [1] and [7] specific epistemic logics, collectively referred to as BAN logics, were proposed to deal with authentication issues. We propose an approach using a combination of temporal and epistemic logics.

By combining both temporal and epistemic logics, we provide a logical framework in which systems requiring both dynamic aspects and informational aspects relating to knowledge can be described. This is particularly important in security protocols, where one wants to ensure that certain knowledge is obtained over time or, at least, that ignorance of potential intruders persists over the whole run of the protocol. These logics have the advantages of a well-defined semantics, an existing body of theoretical work relating to, for example, axiomatisations and complexity, see for example [9], and sound and complete proof methods for example [2].

In this paper, we bring together specification using temporal logics of knowledge and verification using clausal resolution, and apply these to the problem of formally analysing security protocols. In order to show how such protocols can be specified and verified, we consider one very well known protocol, namely the Needham-Schroeder protocol [12]. This protocol has been widely studied with particular problems uncovered via formal analysis, for example [11]. Our aim is to demonstrate the suitability of $KL_{(n)}$, with its resolution method, for security and authentication, rather than bringing new insights to the Needham Schroeder protocol.

Note that, due to lack of space, we will neither give a full description of the resolution calculus, nor full details of the Needham-Schroeder proofs. These details can be found in a companion technical report [4].

2. The Needham-Schroeder Protocol (NSP)

The Needham-Schroeder protocol (NSP) with public keys [12] intends to establish authentication between an agent A who initiates the protocol and an agent B who responds to A. The complete protocol consists of seven messages, but we here focus on a simplified version consisting of only three messages. The messages that we omit are those whereby the agents request other agent's public keys from a server. The protocol can then be described as the following steps:

Message	Direction	Contents
Message 1	$A \rightarrow B$:	$\{N_A, A\}_{pub\_key(B)}$
Message 2	$B \rightarrow A$:	$\{N_B, N_A\}_{pub\_key(A)}$
Message 3	$A \rightarrow B$:	$\{N_B\}_{pub\_key(B)}$

Here $X \rightarrow Y$ denotes that agent X sends agent Y a message. Message contents of the form $\{X,Y\}_{pub\_key(Z)}$ represent messages containing both X and Y but then encrypted with Z's public key. Elements of the form N_X are special items of data, called *nonces*. Typically, agents in the protocol will generate their own unique nonce (often encrypted) which is initially unknown to all other agents.

3. Temporal Logic of Knowledge

The logic, $KL_{(n)}$, a *temporal logic of knowledge* is the fusion of propositional linear-time temporal logic with multi-modal S5. The temporal component is interpreted over a discrete linear model of time with finite past and infinite future and the each modal relation is an equivalence class. This logic has been studied in detail [9] and is the most commonly used temporal logic of knowledge. We use the usual set of operators including $\bigcirc$ (*next*), $\diamondsuit$ (*sometime* or *eventually*), $\square$ (*always*), K_i for *knowledge* and allow an operator **start** to denote the initial moment in time. For details of the syntax and semantics of $KL_{(n)}$ see for example [2].

To prove properties of our specification we use a resolution calculus for $KL_{(n)}$. Due to lack of space we omit the details of the proof method but refer the interested reader to [2, 3].

4. Specifying the NSP in $KL_{(n)}$

In this section, we will use $KL_{(n)}$ to specify the NSP. In particular, we will provide axioms describing the key aspects of both the system and the protocol. In order to do this we use the following syntactic conventions. Let M_1 and M_2 be variables over messages, *Key* be a variable over keys and $X, Y, \ldots$ be variables over agents. Moreover, for every agent, X, we assume there are keys $pub\_key(X)$ and $priv\_key(X)$, while in this protocol A and B are constants representing two specific agents and we introduce an agent C to represent a potential intruder. We identify the following predicates:

- $send(X, Msg, Key)$ (respectively $rcv(X, Msg, Key)$) is satisfied if agent X sends (respectively receives) message Msg encrypted by Key;
- $Msg(M_1)$ is satisfied if M_1 is a message;
- $val\_pub\_key(X,V)$ (respectively $val\_priv\_key(X,V)$) is satisfied if the value of the public (respectively private) key of X is V
- $val\_nonce(N_X, V)$ is satisfied if the value of nonce N_X is V;
- $contains(M_1, M_2)$ is satisfied if the message M_2 is contained within M_1.

To simplify the description, we allow quantification and equality over finite sets of agents, messages and keys; thus, this logic remains essentially propositional.

Specifying Structural Assumptions We begin with various structural assumptions concerning keys and message contents. These are given in Figure 1.

1. $\forall X, Key, M_1.\ send(X, M_1, Key) \Rightarrow \neg contains(M_1, priv\_key(X))$
   — agents will not reveal their private key to others
2. $\forall X, V_1, V_2, V_3.$ $[val\_pub\_key(X,V_1) \Leftrightarrow \square val\_pub\_key(X,V_1)] \wedge [val\_priv\_key(X,V_2) \Leftrightarrow \square val\_priv\_key(X,V_2)] \wedge [val\_nonce(X,V_3) \Leftrightarrow \square val\_nonce(X,V_3)]$
   — the public keys, private keys and nonces of all the agents remain the same during the protocol
3. $\forall X, Y, V.\ (val\_pub\_key(X,V) \wedge val\_pub\_key(Y,V)) \Rightarrow X = Y)$
   — no two agents have the same public keys
4. $\forall Key, M_1.(send(A, M_1, Key) \wedge [contains(M_1, N_A) \vee contains(M_1, N_B)]) \Rightarrow (Key = pub\_key(B))$
   — if agent A sends out messages containing N_A or N_B they must be encrypted with B's public key.
5. $\forall Key, M_2.\ (send(B, M_2, Key) \wedge [contains(M_2, N_A) \vee contains(M_2, N_B)]) \Rightarrow (Key = pub\_key(A))$
   — if agent B sends out messages containing N_A or N_B they must be encrypted with A's public key.

Figure 1. Specifying Structural Assumptions.

Specifying Scenario Assumptions In Figure 2 we instantiate message contents, keys and names for this particular scenario.

Specifying Basic Knowledge Axioms In Figure 3 we specify the attributes of an agent's knowledge.

Specifying Communication Axioms We now specify communication between agents, and how this affects the agent's knowledge. For convenience, we use past-time temporal operators, in particular "$\circledcirc$", meaning in the previous moment in time, and "$\blacklozenge$", meaning at some time in the past. These operators have the usual semantics (see for example [4]). This is shown in Figure 4.

14. $\forall X, M_1, N_1 \ \bigcirc((Msg(M_1) \wedge contains(M_1, N_1)) \Rightarrow (\exists V_1 K_X val\_nonce(N_1, V_1) \ \Leftrightarrow$
$\circledcirc[K_X val\_nonce(N_1, V_1) \vee (\exists Y. \exists V.\ rcv(X, M_1, pub\_key(Y)) \wedge K_X val\_priv\_key(Y, V))]))$
— for all moments except the first moment if M_1 is a message which contains N_1 an agent knows the content of N_1 either if it already knew the content of N_1, or if it received an encrypted version of M_1 that it could decode.

15. $\forall X, Key, M_1.\ rcv(X, M_1, Key) \ \Rightarrow \ \exists Y.\ \blacklozenge\, send(Y, M_1, Key)$
— if an agent receives a message, then there was some agent that previously sent that message

16. $\forall X, Key, M_1, N_1 \ (send(X, M_1, Key) \wedge contains(M_1, N_1) \ \Rightarrow \ \exists V_1.\ K_X val\_nonce(N_1, V_1) \vee \blacklozenge\, rcv(X, M_1, Key)$
— if an agent sends a message M_1 encrypted with *Key*, then it must either know the contents M_1 or just be forwarding the encrypted message as a whole

Figure 4. Specifying Communication Axioms.

6. $\forall M_1$
$Msg(M_1) \Leftrightarrow ((M_1 = m_1) \vee (M_1 = m_2) \vee (M_1 = m_3))$
— in this particular scenario, we just use three messages, m_1,m_2 and m_3. Other (dummy) messages can be added to make this axiom more realistic.

7. $\forall X, Y, Z.$
$(contains(m_1, X) \Leftrightarrow ((X = A) \vee (X = N_A))) \wedge$
$(contains(m_2, Y) \Leftrightarrow ((Y = N_A) \vee (Y = N_B))) \wedge$
$(contains(m_3, Z) \Leftrightarrow (Z = N_B))$
— message m_1 contains only N_A and A, message m_2 contains only N_B and N_A and message m_3 contains only N_B

8. **start** $\Rightarrow$
$val\_priv\_key(A, a_v) \wedge val\_priv\_key(B, b_v) \wedge$
$val\_priv\_key(C, c_v) \wedge val\_pub\_key(A, a) \wedge$
$val\_pub\_key(B, b) \wedge val\_pub\_key(C, c)) \wedge$
$val\_nonce(N_A, a_n) \wedge val\_nonce(N_B, b_n) \wedge$
$val\_nonce(N_C, c_n)$
— the initial values of public and private keys and nonces (we also have negated statements eg **start** $\Rightarrow \neg val\_priv\_key(A, b_v)$ etc).

Figure 2. Specifying Scenario Assumptions.

9. **start** $\Rightarrow \forall X. (\exists V.\ K_X val\_nonce(N_X, V)) \wedge$
$[\forall Y, Z.\ (Y \neq X) \Rightarrow \neg K_Y val\_nonce(N_X, Z)]$
— initially agents only know their own nonces.

10. $\forall X, Y.\ (\exists V. K_X val\_priv\_key(Y, V) \Leftrightarrow (X = Y))$
— agents only know their own private keys

11. $\forall X.\ K_X val\_pub\_key(A, a) \wedge K_X val\_pub\_key(B, b)$
$\wedge K_X val\_pub\_key(C, c)$
— all agents know all the public keys.

12. $\forall X, N, V.\ K_X val\_nonce(N, V) \ \Rightarrow \ \bigcirc K_X val\_nonce(N, V)$
— agents never forget nonces they know

13. $\forall X, Y, V.\ K_X val\_priv\_key(Y, V) \Rightarrow$
$\bigcirc K_X val\_priv\_key(Y, V)$
— agents never forget private keys they know

Figure 3. Specifying Knowledge Axioms.

5. Verifying Properties of the Specification

Once we have the above axioms relating to the specific scenario, we can attempt to prove various statements. Proof is by clausal resolution. For more details see [4].

B's Knowledge on Receipt of N_A The first example will capture the statement "*once B receives the nonce of A encoded by B's public key then B knows the nonce of A*". This can be translated into $KL_{(n)}$ as

$$\Box(rcv(B, m1, pub\_key(B)) \Rightarrow \bigcirc K_B val\_nonce(N_A, a_n))$$

C's Ignorance A key part of this protocol is that information is transferred between agents A and B without agent C ever being able to intercept sensitive information. We can verify this by showing that, in the scenario above, C will never know the value of A's nonce, i.e.

$$\forall V.\ \Box\neg K_C val\_nonce(N_A, V)$$

Confirmation of B**'s Knowledge** Once A receives m_2 (which, in turn, contains N_A) back, then it can infer that B knows the value of N_A, i.e.

$$rcv(A, m_2, pub\_key(A)) \Rightarrow \bigcirc K_A K_B val\_nonce(N_A, a_n)$$

6. Related Work and Conclusions

BAN logics such [1] and [7] analyse security protocols by reasoning about the beliefs of principals. In our approach we use a well studied non-classical logic, i.e. the temporal logic of knowledge to specify and verify protocols. We initially choose to reason about knowledge rather than belief as the epistemic logic of knowledge (S5) is stronger than that of belief (KD45) requiring the axiom $Kl \Rightarrow l$ i.e. if an agent knows l then l is true. We could also easily incorporate beliefs to capture situations when a principal believed items that may not be true.

Further, we use temporal operators to capture temporal information, relating to the order of events for example sending and receiving messages, that is not explicitly stated in BAN logics. In [14], an extension of BAN, time is incorporated in by allowing the past time temporal operators *always in the past* and its dual *sometime in the past* to impose some order on events. We allow a much richer temporal language. BAN logics implicitly assume that beliefs cannot decrease over time. Here we must explicitly state what knowledge persists.

In an approach similar to ours, in [6] a simple branching time logic allowing limited combinations of temporal operators is combined with modal logics of knowledge and permission/obligation to specify security protocols. However no proof method is provided for the resulting logic.

In [8] the authors use Lamport's Raw Temporal Logic of Actions (RTLA) [10] to specify and verify security protocols. In RTLA an action is a statement about pairs of states. Axioms, for example relating to sending and receiving messages, are written with respect to the relevant changes in state. The language allows connectives from classical logic as well as the temporal connectives $\Box$ and $\Diamond$ and other constructs.

Another approach that does use theorem proving, though not particularly related to our approach, is presented in [5]. This paper shows how to introduce time into the Communicating Sequential Processes (CSP) protocol verification framework of [13]. Then CSP is embedded in the PVS (Prototype Verification System).

In this paper, we have shown how temporal logics of knowledge are useful for specifying complex aspects of security protocols. One of the advantages of using a standard combination of temporal and modal logics is that there is a clear semantics for this logic. This was a problem with the early BAN logics. Secondly there is an existing bank of work relating to axiomatisations, complexity, proof methods etc that can be applied. In combination with clausal resolution techniques we have developed, this allows us to carry out verification of properties of security protocols. While there has been work on verification of such protocols before, the clarity of the logic, together with the flexibility of the proof technique, makes this work important. In the future, we will consider adding first-order aspects to the logic, thus allowing the verification of infinite state protocols.

Acknowledgements This work was partially supported by EPSRC research grant GR/R45376/01.

References

[1] M. Burrows, M. Abadi, and R. Needham. A Logic for Authentication. In *Proceedings of the Royal Society of London*, volume 426, pages 233–271, 1989.

[2] C. Dixon and M. Fisher. Resolution-Based Proof for Multi-Modal Temporal Logics of Knowledge. In S. Goodwin and A. Trudel, editors, *Proceedings of TIME-00 the Seventh International Workshop on Temporal Representation and Reasoning*, Cape Breton, Nova Scotia, Canada, July 2000. IEEE Press.

[3] C. Dixon, M. Fisher, and M. Wooldridge. Resolution for Temporal Logics of Knowledge. *Journal of Logic and Computation*, 8(3):345–372, 1998.

[4] C. Dixon, M.-C. F. Gago, M. M. Fisher, and W. van der Hoek. Using temporal logics of knowledge in the formal verification of security protocols. Technical Report ULCS-03-022, University of Liverpool, Department of Computer Science, 2003. `http://www.csc.liv.ac.uk/research/techreports`.

[5] N. Evans and S. Schneider. Analysing time dependent security properties in CSP using PVS. In *ESORICS*, pages 222–237, 2000.

[6] J. Glasgow, G. MacEwen, and P.Panangaden. A Logic to Reason About Security. *ACM Transactions on Computer Systems*, 10(3):226–264, August 1992.

[7] L. Gong, R. Needham, and R. Yahalom. Reasoning about Belief in Cryptographic Protocols. In *Proceedings of the IEEE Computer Societey Symposium on Research in Security and Privacy*, pages 234–248. IEEE Computer Society Press, 1990.

[8] J. W. Gray and J. McLean. Using Temporal Logic to Specify and Verify Cryptographic Protocols. *Computer Security Foundations Workshop*, 12:108–116, 1995.

[9] J. Y. Halpern and M. Y. Vardi. The Complexity of Reasoning about Knowledge and Time. I Lower Bounds. *Journal of Computer and System Sciences*, 38:195–237, 1989.

[10] L. Lamport. The Temporal Logic of Actions. Technical Report Research Report 79, DEC Systems Research Center, Palo Alto, CA, 1991.

[11] G. Lowe. Breaking and Fixing the Needham-Schroeder Public-key Protocol Using csp and fdr. In T. Margaria and B. Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems: second international workshop, TACAS '96*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Spinger, 1996.

[12] R. Needham and M. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM*, 21:993–999, 1978.

[13] S. Schneider. Verifying authentication protocols with CSP. In *PCSFW: Proceedings of The 10th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1997.

[14] P. Syverson. Adding Time to a Logic of Authentication. In *Proceedings of the First ACM Conference on Computer and Communications Security*, pages 97–101. ACM Press, 1993.

An Encapsulated Authentication Logic for Reasoning About Key Distribution Protocols[*]

Iliano Cervesato
Tulane University
iliano@math.tulane.edu

Catherine Meadows
Naval Research Laboratory
meadows@itd.nrl.navy.mil

Dusko Pavlovic
Kestrel Institute
dusko@kestrel.edu

Abstract

Authentication and secrecy properties are proved by very different methods: the former by local reasoning, leading to matching knowledge of all principals about the order of their actions, the latter by global reasoning towards the impossibility of knowledge of some data. Hence, proofs conceptually decompose in two parts, each encapsulating the other as an assumption. From this observation, we develop a simple logic of authentication that encapsulates secrecy requirements as assumptions. We apply it within the derivational framework to derive a large class of key distribution protocols based on the authentication properties of their components.

1 Introduction

Secrecy and authentication are the two main properties guaranteed by cryptographic protocols. Since they are highly dependent upon each other, they are commonly intertwined in protocol analysis. Indeed, early attempts to develop logics that reason only about authentication, such as BAN logic [2], were believed to be doomed to failure because the need to draw unverifiable conclusions about secrecy from authentication guarantees led to confusion and misunderstanding. It caused such problems as Lowe's attack [9] on the Needham-Schroeder public key protocol [12], which BAN had proved secure.

On the other hand, restricting one's reasoning to authentication alone has many benefits, if only it can be achieved. Authenticity properties specify conditions about the order of actions [7, 10]: if a particular action occurs, then some other set of actions must have occurred before it, and in a particular (partial) order. Thus, it is conceivable that a useful logic for protocol analysis could be developed that allows using a minimal set of primitives describing principal actions (e.g., sending and receiving messages and generating nonce, etc.) and reasoning about their order.

In this paper we present a logic that does just that. Unlike BAN logic, we do not avoid reasoning about secrecy entirely. Rather, we encapsulate the secrecy properties needed as proof obligations that can be proved by other means (we are developing a companion logic of secrecy to do just that). Our authentication logic reasons about the partial order of actions, Lamport-style. When a consequence of secrecy is needed, we include an assumption that defines a proof obligation which can be thought of as a system call to another verification method.

The logic has much in common with the compositional logic described in [5, 8]. Like that logic, it supports deriving complex protocols from simple ones using composition, refinement, and transformation. However, unlike that logic, in which predicates are defined as they are needed, the new logic relies on just three built-in predicates describing the ordering of events. It allows inferring the actions of other principals and their order based on one's local observations only: a fairly simple deductive process, even in our setting. It does not establish secrecy results, which use an entirely different flavor of logical inference: a form of inductive reasoning over the global knowledge of all principals. The clean separation of authentication proofs and secrecy proofs results in simple and elegant analyses.

We have applied an earlier version of this logic to the Group Domain of Interpretation (GDOI) Protocol [11]. Its simplicity turned out to be quite helpful in allowing us to identify a security flaw that careful reviews and another formal analysis had previously missed. The simplicity of the proof system allowed us to identify a key assumption being made that was not consistent with the protocol design. When the assumption was removed, the flaw was revealed.

We apply this logic to study the general mechanism of key distribution from an authentication perspective. Key distribution protocols feature complex interaction of secrecy and authentication requirements, and are therefore a prime target for our methodology. We start from an

[*]Supported by ONR and NSF.

abstract two-party key distribution protocol and, through refinements and transformations, we systematically derive the skeleton of such well-known protocols as shared-key Needham-Schroeder (NSSK) [12, 13], Denning-Sacco [6], and Kerberos 4 and 5 [14, 15]. An overview of the resulting taxonomy is displayed in Figure 1.

The main contribution of this work is twofold: (1) We formally separate the reasoning about authenticity from the reasoning about secrecy, and develop a simple logic of partial orders that supports the former. The secrecy inferences present in every proof of authentication are encapsulated as assumptions. (2) We use this logic within the derivational framework [5, 8] to perform a systematic analysis of an important branch of the family of key distribution protocols. This yields a novel classification of these protocols based on the mechanisms used to construct them and the properties they support. This promotes a deeper understanding of these fundamental protocols than individual analyses, and is readily extensible as new members are proposed.

This work is organized as follows: in Section 2 we explain our authentication logic. We use it to express the basic key distribution mechanism in Section 3. We extend it in the direction of NSSK in Section 4 with nonce-based recency and key confirmation. We extend it in a different direction with timestamp-based recency in section 5 obtaining the Denning-Sacco protocol as well as Kerberos 4 and 5. Section 6 concludes with statements of future work.

2 A Logic of Pure Authentication

As a principal A executes a protocol P, the events she observes locally (receiving a messages, comparing a component with an expected value, etc) allow her to make deductions about the actions of the principals she is interacting with. This implicitly identifies a class $\mathcal{Q}_A$ of possible runs, each of which intersperses her own actions with compatible actions by the other participants. As an authentication property Prop also identifies a class $\mathcal{Q}_{\mathsf{Prop}}$ of legal runs for P, the verification task traditionally reduces to showing that $\mathcal{Q}_A$ is contained in $\mathcal{Q}_{\mathsf{Prop}}$, and similarly for the other parties in the protocol. Every run in $\mathcal{Q}_A$ but not in $\mathcal{Q}_{\mathsf{Prop}}$ is an attack on A with respect to Prop.

We take a different approach: rather than comparing $\mathcal{Q}_A$ with the legal runs of a given authentication property, we synthesize a logical expression Φ describing $\mathcal{Q}_A$. This explicit representation is carefully engineered to be compositional: we dissect A's observations into elementary components and give a logical representation of the property they each realize (their *raison d'être* in a protocol). We similarly give a logical justification of the various mechanisms that allow combining components into bigger protocol fragments, and in particular of what properties emerge from the properties of the parts. By iterating this process all the way to A's original observations, we derive a formula, Φ, that in a strong sense describes *the* authentication properties of $\mathcal{Q}_A$. Indeed, this constructions provides us with a clear view of the properties contributed by each component and whether they propagate to Φ. We often restrict our attention to interesting scenarios by assuming, for example, that other principals behave honestly, or that a certain key has not been compromised. These assumptions are elective.

Rather than checking that a protocol satisfies a given property Prop, our approach enumerates the properties supported by a protocol based on its construction. Whenever an expected property is not manifested, we can rapidly point to a missing component or a composition mechanism failing to propagate it, and produce a counterexample, as done in [11]. We can also scrutinize the formula Φ summarizing the possible runs of each principal A in the light of a well-known authentication property, such as matching histories [7] or agreement [10].

2.1 Specifying Protocols

We begin by presenting a syntax to describe security protocols and enough of its execution semantics to define the notion of observation, a central concept in this work. The interested reader will find further details, as well as a semantics of protocol execution, in [4].

We use the letters A, B, and S to denote the *principals* participating in a given protocol. X, Y, ... will be variables ranging over principals. Principals exchange *messages*. This is modeled as an abstract term algebra $\mathcal{T}$ over a set of variables, constants and operators. Principals are a subclass of terms, and so are standard classes in crypto-protocols such as nonces, keys and timestamps. We write m for generic message, but use k, n, and t, for keys, nonces and timestamps. The letters $x, y, z, \ldots$ denote term variables. In this work, we will make use of two operators: (m, m') for the concatenation of m and m', and $(k\ m)$ for the encryption of m with shared key k. $\mathcal{T}$ may contain additional constructors, but we will not need them here.

Principals participate in a protocol by performing atomic *actions*. The actions we will rely on are summarized in the following table:

Action	*Form*	*Informal meaning*
send	$\langle m : A \rightarrow B \rangle$	The term m is sent, purportedly from A to B
receive	$(x : Y \rightarrow Z)$	Term, source and destination are received into x, Y and Z
match	$(m/p(\vec{x}))$	Term m is matched against term $p(\vec{x})$, binding $\vec{x}$
new	$(\nu\ x)$	A fresh value is created and stored in variable x
now	$(\tau\ x)$	The system time is read and stored in variable x

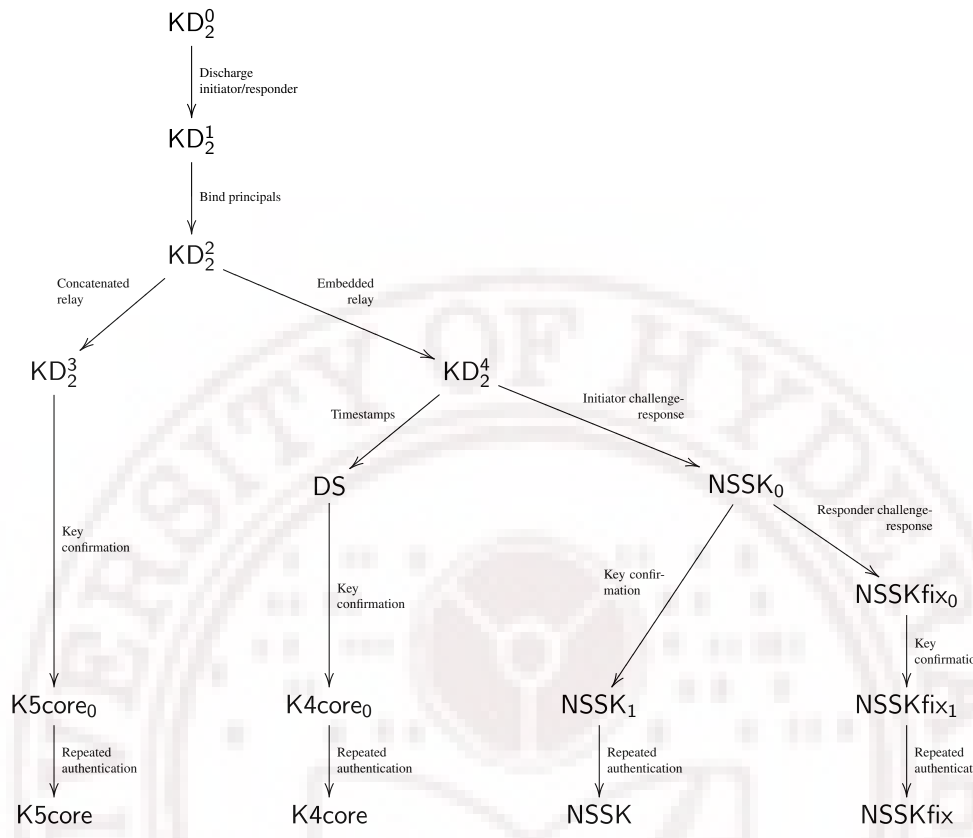


Figure 1. Overview of the Derivation of Key-Distribution Protocols

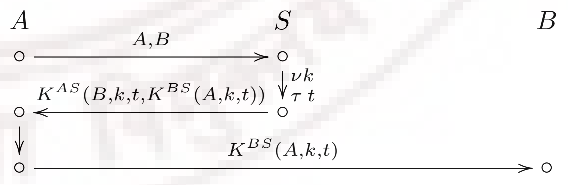
The variables x, Y, Z in **receive**, $\vec{x}$ in **match**, and x in **new** and **now** are binding occurrences: any subsequent mention in an expression involving actions are interpreted as bound by them. The semantics of actions is formalized in [4]. We will often use partial descriptions of actions, and elide e.g., the source and the destination, as in $\langle m \rangle$ or (y), and merge a reception and subsequent matches, writing (m).

A *role* is the complete code that a principal executes on her host to engage in a given protocol. We model a role as a collection of actions performed by a principal. We allow actions to be composed either sequentially (using ";" as a role constructor) or concurrently (using "$\otimes$"). The free variables of a role are its parameters, and should be instantiated prior to execution. The principal executing the role is a distinguished parameter. A *protocol* is a collection of roles. For example, the Denning-Sacco protocol [6] is specified as follows:

$$\begin{aligned}
\mathsf{DS\_server}[S] &= (A, B : A \to S) \; ; \; (\nu\, k \;\otimes\; \tau\, t) \; ; \\
&\quad\ \langle K^{AS}(B, k, t, K^{BS}(A, k, t)) : S \to A \rangle \\
\mathsf{DS\_init}[A; S, B] &= (K^{AS}(B, k, t, M) : S \to A) \; ; \\
&\quad\ \langle A, B : A \to S \rangle \; ; \; \langle M : A \to B \rangle \\
\mathsf{DS\_resp}[B; S] &= (K^{BS}(A, k, t) : A \to B)
\end{aligned}$$

We will come back to this protocol in Section 5.

We will be primarily interested in the trace or *run* of a (possibly partial) execution, i.e., the set of actions executed by each principal and their relative ordering. For example the expected run of the Denning Sacco protocol is given as follows:



where we have liberally compacted our notation for succinctness. A run is a partial order over the set of *events* of the form a_A, where a is an action and A is the principal who has executed it. A *valid run* is subject to two conditions: (1) every receive event is preceded by a send in the partial order; (2) every match is successful. Given a valid run Q, the *local observation* of a principal A, denoted Q_A, is the restriction of Q to just the actions performed by A, and their relative order.

2.2 Reasoning about Authentication

We now set up a logic to draw inferences about valid runs. It will allow us to reconstruct the runs compatible with a principal's observations, often under assumptions.

We consider an instance of first-order logic with just three predicate forms:

a	*Event a has occurred*
$a < b$	*Event a has occurred before event b*
$a = b$	*a and b are the same event*

A formula combines these atomic predicates by means of the traditional connectives and quantifiers of first-order logic. Within an event, we omit the intended sender and recipient in a send or receive action when unimportant or easily reconstructible from the context. Additional abbreviations will make our discussion more clear and succinct:

This ...	... abbreviates ...
$(p)_A$	$(x)_A < (x/p)_A$
$(\!(p)\!)_A$	$(x)_A < (x/p')_A$ for p subterm of p'
$\langle\!\langle m\rangle\!\rangle_A$	$\langle m'\rangle_A$ for m subterm of m'
$\langle m\rangle_{A<}$	$\exists a = \langle m\rangle_A \wedge \forall b = \langle\!\langle m\rangle\!\rangle_B .\ a \leq b$
$\langle\!\langle m\rangle\!\rangle_{A<}$	$\exists a = \langle\!\langle m\rangle\!\rangle_A \wedge \forall b = \langle\!\langle m\rangle\!\rangle_B .\ a \leq b$
$a \prec b$	$b \Rightarrow a < b$

A valid run Q is immediately described by a formula Φ_Q consisting of a conjunction of the first two types of predicates above. An observation Q_A is similarly mapped to a formula Φ_A. Dually, an arbitrary formula Φ can be validated against a run Q, realizing a form of model checking [4].

In the sequel, we will define logical tools to complete the local observation Q_A of a principal A into a compatible run Q (which may or may not be the expected run). We will do so by building a tautology $\Phi_A \Rightarrow \Phi_Q$ which we abbreviate as $A : \Phi_Q$. Interesting completions will often require making assumptions Ψ, so that the general form of our statements will be $A : \Psi \Rightarrow \Phi_Q$.

2.3 Honesty and Secrecy Assumptions

We will often need a principal A deducing $A : \Phi$ to assume that another principal B is *honest* in order to draw interesting conclusions. By this, we mean that B does not deviate from his assigned role: if A ascertains that B has executed any action a_B in his role, she can be assured that he has executed all the actions preceeding a, or concurrent with it. The *honesty assumption* of the server of the Denning-Sacco protocol is as follows:

$$(A,B)_S \prec \begin{bmatrix}(\nu\, k)_S \\ (\tau\, t)_S\end{bmatrix} \prec \langle K^{AS}(B,k,t,K^{BS}(k,A,t))\rangle_S$$

We abbreviate it as $\mathsf{honest}\ S$. We have extensively studied this notion in previous work [5, 11].

The *secrecy assumption* is novel: it allows specifying that certain keys have not been compromised. A shared key k is uncompromised for a group G of agents if the only principals that can perform an encryption or a decryption using k are the members of G. In symbols,

$$\begin{aligned}\mathsf{uncomp}(k,G) \quad \triangleq \quad & \langle\!\langle k\, m\rangle\!\rangle_{X<} \ \Rightarrow\ X \in G \\ \wedge\ & (x/k\, y)_X \ \ \Rightarrow\ X \in G\end{aligned}$$

Notice that the body of this definition expresses the semantics of shared-key cryptography: the first line says that only members of G can produce an encryption using k and send it in a message, the second says that only these principals can use the pattern $(k\ y)$ to access the contents of a term encrypted with k. Notice also that this expression defines the binding between a key and the principals who can use it.

uncomp acts as an interface between the logic of pure authentication developed in this paper, and the logic of secrecy which will be the subject of a sequel to this work. Here, we use it only as an *assumption*. There, we will be able to *prove* formulas of the form $\mathsf{uncomp}(k,G)$, which will permit discharging the assumption $\mathsf{uncomp}(k,G)$. This combination of logics will be particularly useful for studying staged protocols such as Kerberos, where a key is distributed for the purpose to protecting another key.

2.4 Axioms

We now describe some of the logical tools that allow extending a local observation into a compatible run. Most of these ideas have been extensively discussed elsewhere [5, 11], in which case we keep the presentation brief.

The *freshness axiom* (new) describes the behavior of the $(\nu\, n)$ action in logical terms:

$$\begin{aligned}(\nu\, n)_B \wedge a_A \Rightarrow (n \in FV(a)\ & \Rightarrow (\nu\, n)_B < a_A \\ & \wedge (A \neq B \Rightarrow (\nu\, n)_B < \langle\!\langle n\rangle\!\rangle_B < (\!(n)\!)_A \leq a_A))\end{aligned}$$

The first part says that ν is a binder, that is, any event a mentioning n necessarily occurs *after* $(\nu\ n)$. The second line requires that if the agent B executing $(\nu\ n)$ and the principal A executing a are different, then B must have used a send action to transmit n and A must have acquired it by means of a receive action.

The *receive axiom* (rcv) says that everything that is received must have been originated by someone:

$$A :\ (\!(m)\!)_A \Rightarrow\ \exists X.\ \langle\!\langle m\rangle\!\rangle_{X<} < (\!(m)\!)_A$$

The extensively studied *challenge-response axiom schema* (cr) [5, 11] abstractly describes the central concept of nonce-based challenge-response:

$$\begin{aligned}& A : \Phi' \ \wedge\ (\nu n)_A < \langle\!\langle c^{AX} n\rangle\!\rangle_{A<} < (\!(r^{AX} n)\!)_A \\ & \Rightarrow\ (\nu n)_A < \langle\!\langle c^{AX} n\rangle\!\rangle_{A<} < (\!(c^{AX} n)\!)_X < \langle\!\langle r^{AX} n\rangle\!\rangle_{X<} < (\!(r^{AX} n)\!)_A\end{aligned}$$

where c^{AX} is the challenge structure issued by A, r^{AX} is the corresponding response originated by X, and Φ' represents some additional precondition, usually an honesty or uncomp assumption.

One instance of (cr) that we will use in the sequel has c^{AX} be the identity (the nonce is sent in the clear), the response the encryption of the nonce with a key K^{AX} shared between A and X, and Φ' requiring K^{AX} not to be compromised for A and X. We obtain

$$A : \mathsf{uncomp}(K^{AX}, [A, X]) \wedge (\nu n)_A < \langle\langle n \rangle\rangle_{A<} < ((K^{AX} n))_A$$
$$\Rightarrow\ (\nu n)_A < \langle\langle n \rangle\rangle_{A<} < ((n))_X < \langle\langle K^{AX}\ n \rangle\rangle_{X<} < ((K^{AX}\ n))_A$$

A proof of this instance of (cr) goes as follows: starting from A's own observations (the first line above), axiom (rcv) entails that some agent Y has originated $\langle\langle K^{AX}\ n \rangle\rangle$. By the uncomp assumption, Y must be either X or A; the latter possibility is excluded since no such actions occurs among A's observations. Axiom (new) completes the second line by sandwiching X's reception of n between A's transmission of the nonce and X's issuing of $\langle\langle K^{AX}\ n \rangle\rangle$.

The novel *timestamp axiom* (ts) describes the semantics of timestamps.

$$A : \mathsf{honest}\ X\ \ \wedge \langle\langle t \rangle\rangle_{X<} < ((t))_A$$
$$\Rightarrow\ (\underline{\tau}\ t)_A < (\tau t)_X < \langle\langle t \rangle\rangle_{X<} < ((t))_A < (\overline{\tau}\ t)_A$$

The antecedent of this formula assumes that A receives a message containing an acceptable timestamp t, and she has the certainty that an honest X has originated $\langle\langle t \rangle\rangle$. Given these hypotheses, she can deduce that X had indeed looked up t and sent it out, and that these actions took place within what she regards as the window of validity of this timestamp. Here, $(\underline{\tau}\ t)_A$ is the earliest point in time where A would accept t as valid, and $(\overline{\tau}\ t)_A$ is the dual upperbound. They are events internal to A representing time points calculated from t by considering what she deems as acceptable clock skews and network delays. What is important here is that they bound X's actions by events under A's control. In the sequel, we will discharge the assumption that A is certain that X has sent this timestamp whenever the message is authenticated.

Except for (new), all the axioms in this section are instances of the *send-receive axiom schema* (sr):

$$A\ :\ \exists X.\forall\vec{y}.\ ((f^{AX}(\vec{y})))_A\ \wedge\ \Phi(X, \vec{y})$$
$$\Rightarrow\ \langle\langle f^{AX}(\vec{y}) \rangle\rangle_{X<}\ <\ ((f^{AX}(\vec{y})))_A\ \wedge\ \Psi(X, \vec{y})$$

It says that A knows that, for some principal X, the message structure f^{AX} assures that, if she receives a message containing $f^{AX}(\vec{y})$, where X and $\vec{y}$ satisfy some precondition Φ, then X must have originated $f^{AX}(\vec{y})$, and that X and $\vec{y}$ satisfy some postcondition Ψ.

2.5 The Derivational Approach

Axioms suffice to extend a local observation into all of its compatible runs. Adopting a technique developed in previous work [5, 11], a more effective way to study a large protocol is to decompose it into elementary exchanges such as challenge-response and basic key distribution (see Section 3), derive the runs compatible with observations for these parts, and then reassemble them into formulas describing the compatible runs of the overall protocol. The logical tools that permit doing this are called refinements and transformations. A *refinement* reflects changes within one or more messages in a protocol into the formulas describing the runs compatible with an upgraded observation. For example, inserting a timestamp in a message has the effect of strengthening the recipient's guarantees. A *transformation* is similar but supports alterations to the exchange pattern of the protocol. For example, extending a protocol with an additional round of challenge-response is a transformation.

Besides the appeal of a divide-and-conquer methodology, the *derivational approach*, as this technique is known, supports reusing component-formulas pairs whenever they occur in another protocols. Moreover, the application of refinements and transformations can be made systematic, which gives rise to protocol taxonomies [5]: a rational classification of protocols that not only aids our understanding of these complex objects, but also helps choosing or devising a protocol based on desired features and properties. A tool is under development that will assist us building taxonomies that are much larger than what we have so far been able to construct by hand [1].

A detailed discussion of refinements, transformations and the derivational approach in general would be rather lengthy and technical. The reader is better served by consulting the existing literature [5]. Therefore, we will introduce these powerful tools only informally as we need them in the sequel. Section 5.1 gives some details of a new refinement.

3 Basic Key Distribution

In this section, we use the above logic to study the general mechanism of key distribution from an authentication perspective. Key distribution protocols feature complex interaction of secrecy and authentication requirements, and are therefore a prime target for our methodology. Indeed, their general goal is to authenticate two principals A and B to each other through communications with a server S along pre-established channels protected by secret keys. The distributed key protects future communications between A and B. We analyze the authentication aspects of such protocols, assuming that the keys to the server are not compromised. Proving that the distributed key is secret pertains to our forthcoming logic of secrecy, and is not addressed here.

For reasons of space, we give a complete proof of only the first derivation in this paper. The other proofs are analogous. Other proofs and more detail may be found in [4].

We start with an abstract form of two-party key distribution, which we analyze from scratch, relying on axioms only. Then, through a series of refinements and transformations, we derive the skeleton of several well-known protocols, which we will further build up in the next sections.

In our first and most abstract protocol, KD_2^0, a server S spontaneously generates a key k and distributes it to known principals A and B encrypted with keys K^{AS} and K^{BS} he shares with each of them. The protocol is given by the following roles. The actions of A and B are symmetric at this stage (only A's role is shown).

$$\begin{array}{rcl} \mathsf{KD}_2^0\_\mathsf{server}[S;A;B] & = & \nu\, k\ ; \quad \langle K^{AS}\, k : S \to A\rangle \\ & & \qquad\quad \otimes \langle K^{BS}\, k : S \to B\rangle \\ \mathsf{KD}_2^0\_\mathsf{client}[A;S,B] & = & (K^{AS}\, k : S \to A) \end{array}$$

The key server and the clients are given everybody's name as parameters to their respective roles. The expected run provides a clearer view of this exchange:

$$\begin{tikzcd} A & S & B \\ & \circ \arrow[d, "\nu k"] & \\ \circ & \circ \arrow[l, "K^{AS}\, k"'] \arrow[r, "K^{BS}\, k"] & \circ \end{tikzcd}$$

We will now take the point of view of each principal and infer a formula representing the runs that are compatible with its observations. We start with A (B is symmetric). The only event she observes is $(K^{AS}\ k : S \to A)$. Under the assumptions that K^{AS} is shared only by A and S and the honesty of S (derived from his role), A can reconstruct the expected run. The formal derivation is as follows:

$$\begin{array}{rl} \mathit{Obs} & : (K^{AS}\, k : S \to A)_A \\ (\mathsf{rcv}) & : \langle\!\langle K^{AS}\, k\rangle\!\rangle_{X<} < (K^{AS}\, k : S \to A)_A \\ \mathsf{uncomp} & : X = A \text{ or } X = S \\ \mathit{Obs} & : X \neq A \\ \mathsf{honest}\ S & : (\nu k)_S \prec \begin{bmatrix} \langle K^{AS} k : S \to A\rangle_{S<} \\ \langle K^{BS} k : S \to B\rangle_{S<} \end{bmatrix} \\ \hline & (\nu k)_S < \begin{bmatrix} \langle K^{AS} k : S \to A\rangle_{S<} \\ \langle K^{BS} k : S \to B\rangle_{S<} \end{bmatrix} < (K^{AS} k)_A \end{array}$$

All the proofs in this paper have a similarly simple form. We will omit them for brevity. A compact representation of the overall formula follows:

$$\begin{array}{ll} A: & \mathsf{uncomp}(K^{AS}, [A,S]) \ \wedge\ \mathsf{honest}\ S\ \wedge (K^{AS}\ k)_A \\ \Rightarrow & (\nu\, k)_S < \begin{bmatrix} \langle K^{AS}\ k : S \to A\rangle_{S<} \\ \langle K^{BS}\ k : S \to B\rangle_{S<} \end{bmatrix} < \ (K^{AS}\ k)_A \end{array}$$

The server S does not conclude more than he observes since he is the recipient of no message.

In this protocol, each principal knows the identity of every other party ahead of time. This is not how typical key distribution protocols work: instead, one client, say A, takes the role of *initiator* by sending a message to the server saying she wants to communicate with B (the *responder*). This alteration is formalized by means of the *discharging transformation*, which replaces a parameter in a role with a received value.

We apply this transformation twice to protocol KD_0^2, discharging A and B as parameters to S's role. Before presenting the roles of the resulting protocol, KD_1^2, a glimpse at its expected run will help visualize what we have achieved:

$$\begin{tikzcd} A & S & B \\ \circ \arrow[r, "{A,B}"] & \circ \arrow[d, "\nu k"] & \\ \circ & \circ \arrow[l, "K^{AS}\, k"'] \arrow[r, "K^{BS}\, k"] & \circ \end{tikzcd}$$

The roles of KD_1^2 clearly show that S does not have A and B as parameters any more:

$$\begin{array}{rl} \mathsf{KD}_2^1\_\mathsf{server}[S] & = (A, B : A \to S)\ ;\ \nu\, k\ ; \\ & \quad \langle K^{AS}\, k\!: S \to A\rangle \otimes \langle K^{BS}\, k\!: S \to B\rangle \\ \mathsf{KD}_2^1\_\mathsf{init}[A;S,B] & = \langle A, B : A \to S\rangle\ ;\ (K^{AS}\ k : S \to A) \\ \mathsf{KD}_2^1\_\mathsf{rsp}[B;S,A] & = (K^{BS}\ k : S \to B) \end{array}$$

Observe that the roles of A and B are not symmetric any more. Note also that it would make little difference if A transmitted just "B" as her first message since her name is present in the "from" field of this action.

The properties characterizing A's and B's views are summarized next.

$$\begin{array}{l} A : \mathsf{uncomp}(K^{AS}, [A,S]) \ \wedge\ \mathsf{honest}\ S\ \wedge \\ \qquad\qquad\qquad \langle A, B\rangle_A \qquad\qquad\qquad < (K^{AS}\ k)_A \\ \Rightarrow \begin{bmatrix} \qquad\qquad\quad \langle A, B\rangle_A \\ (A,X)_S < (\nu\, k)_S < \begin{bmatrix} \langle K^{AS}\ k\rangle_{S<} \\ \langle K^{XS}\ k\rangle_{S<} \end{bmatrix} \end{bmatrix} < (K^{AS}\ k)_A \end{array}$$

$$\begin{array}{l} B : \mathsf{uncomp}(K^{BS}, [B,S]) \ \wedge\ \mathsf{honest}\ S\ \wedge (K^{BS}\ k)_B \\ \Rightarrow\ (X,B)_S < (\nu\, k)_S < \begin{bmatrix} \langle K^{XS}\ k\rangle_{S<} \\ \langle K^{BS}\ k\rangle_{S<} \end{bmatrix} < (K^{BS}\ k)_B \end{array}$$

Observe that A has no way to determine whether S transmitted the key k to B or to some other party X. She can only infer that S received a request for a key involving herself and some X, not necessarily B. By a similar argument, B cannot ascertain to whom k was distributed even if A appears among the parameters of his role.

This problem is traditionally solved by having S include B's name into the message directed to A, and A's name into B's message. The *binding refinement* of [5] achieves precisely this effect: it modifies a submessage $k\ m$ (encrypted with an uncompromised key k) into the term $k\,(m, m')$ thereby cryptographically authenticating m' to any party able to access the ciphertext. By applying this refinement twice (once for A and once for B), S can inform A and B of whom it created k for. This also allows us to discharge A as a parameter in B's role. Let KD_2^2 be the resulting protocol. Its expected run is given by the following diagram:

$$\begin{tikzcd}
A & S & B \\
\circ \arrow[r, "{A,B}"] & \circ \arrow[d, "\nu k"] & \\
\circ & \circ \arrow[l, "{K^{AS}\,(B,k)}"'] \arrow[r, "{K^{BS}\,(A,k)}"] & \circ
\end{tikzcd}$$

(We stop showing roles for space reasons.) A and B can derive the following properties, respectively:

$$\begin{aligned}
&A : \mathsf{uncomp}(K^{AS}, [A,S]) \ \wedge\ \mathsf{honest}\ S \ \wedge \\
&\qquad \langle A,B\rangle_A \quad < (K^{AS}\,(B,k))_A \\
&\Rightarrow \begin{bmatrix} \langle A,B\rangle_A \\ (A,B)_S < (\nu\,k)_S < \begin{bmatrix} \langle K^{AS}\,(B,k)\rangle_{S<} \\ \langle K^{BS}\,(A,k)\rangle_{S<} \end{bmatrix} \end{bmatrix} \\
&\qquad < (K^{AS}\,(B,k))_A
\end{aligned}$$

$$\begin{aligned}
&B : \mathsf{uncomp}(K^{BS}, [B,S]) \ \wedge\ \mathsf{honest}\ S \ \wedge (K^{BS}\,(A,k))_B \\
&\Rightarrow\ (A,B)_S < (\nu\,k)_S < \begin{bmatrix} \langle K^{AS}(B,k)\rangle_{S<} \\ \langle K^{BS}(A,k)\rangle_{S<} \end{bmatrix} < (K^{BS}(A,k))_B
\end{aligned}$$

While these formulas are similar to what we derived for protocol KD_2^1, A and B now know that the key k is intended for the two of them to communicate, not a third party (assuming, of course that S is honest and that the keys K^{AS} and K^{BS} are not compromised). This correction becomes crucially important when A and B attempt to use k.

While KD_2^2 achieves a minimal form of key distribution, few actual protocols have this message structure. Indeed, with the exception of recent group protocols [11], nearly all key distribution protocols based on shared keys have the server send both components $K^{AS}\,(B,k)$ and $K^{BS}\,(A,k)$ to one principal, who then forwards the part he does not understand to the other. This intuition is logically harnessed by means of the *relay transformation* which yields the following exchange structure:

$$\begin{tikzcd}
A & S & B \\
\circ \arrow[r, "{A,B}"] & \circ \arrow[d, "\nu k"] & \\
\circ \arrow[d] & \circ \arrow[l, "{K^{AS}\,(B,k),\ K^{BS}\,(A,k)}"'] & \\
\circ \arrow[rr, "{K^{BS}\,(A,k)}"] & & \circ
\end{tikzcd}$$

In the corresponding protocol, which we will call KD_2^3, S concatenates $K^{AS}\,(B,k)$ and $K^{BS}\,(A,k)$, and sends the resulting message to A, who then forwards $K^{BS}\,(A,k)$ to B. Several protocols, e.g., Kerberos 5, follow this pattern. Note that the component $K^{BS}\,(A,k)$ is opaque to A, so her role mentions a generic message M.

Applying the relay transformation to the formula characterizing A's view in KD_2^2 yields the following expression:

$$\begin{aligned}
&A : \mathsf{uncomp}(K^{AS}, [A,S]) \ \wedge\ \mathsf{honest}\ S \ \wedge \\
&\quad \langle A,B\rangle_A < (K^{AS}\,(B,k),\ M)_A < \langle M\rangle_A \\
&\Rightarrow \begin{bmatrix} \langle A,B\rangle_A \\ (A,B)_S < (\nu\,k)_S < \langle K^{AS}\,(B,k), \boxed{K^{BS}\,(A,k)}\rangle_{\boxed{S<}} \end{bmatrix} \le \\
&\quad \le \langle K^{AS}\,(B,k), \boxed{M}\rangle_{\boxed{X<}} < (K^{AS}\,(B,k), \boxed{M})_A < \langle M\rangle_A
\end{aligned}$$

Compared to the analogous property of KD_2^2, A's receive action contains a generic M, and the server sends a concatenated message rather than the two components separately. This has two implications, highlighted in the boxes:

1. While, by the honesty assumption, A knows that S has sent $K^{AS}\,(B,k), K^{BS}\,(A,k)$, she cannot ascertain that the generic message M she receives is indeed $K^{BS}\,(A,k)$.

2. Since K^{AS} is uncompromised, A knows that S has originated $K^{AS}\,(B,k)$, but she cannot be sure of who originated the message $K^{AS}\,(B,k), M$ she received: hence the variable X for its originator, and the $\le$ relation, a result of applying axiom rcv. Indeed an attacker could have replaced $K^{BS}(A,k)$ with an arbitrary message in an undetectable way. Such a behavior has been documented for Kerberos 5 [3].

Additionally, observe that A's last send has little bearing on the overall property and could be dropped.

For similar reasons, B has no way to know who forwarded the message he receives.

$$\begin{aligned}
&B : \mathsf{uncomp}(K^{BS}, [B,S]) \ \wedge\ \mathsf{honest}\ S \ \wedge (K^{BS}\,(A,k))_B \\
&\Rightarrow\ (A,B)_S < (\nu\,k)_S < \langle K^{AS}\,(B,k),\ K^{BS}\,(A,k)\rangle_{S<} < \\
&\qquad < \langle K^{BS}\,(A,k)\rangle_{X<} < (K^{BS}\,(A,k))_B
\end{aligned}$$

Note that if B were able to infer that X is indeed A, he could also conclude that A knows the key k.

We conclude this section by deriving a variant of KD_2^3, in which B's component is embedded in A's rather than concatenated with it. Protocols that follow this approach include NSSK, Denning-Sacco and Kerberos 4.

This is formally achieved through a variant of the binding refinement used earlier. Applying it to KD_2^3 yields protocol KD_2^4, which has the following expected run:

$$\begin{tikzcd}
A & S & B \\
\circ \arrow[r, "{A,B}"] & \circ \arrow[d, "\nu k"] & \\
\circ \arrow[d] & \circ \arrow[l, "{K^{AS}\,(B,k,K^{BS}\,(A,k))}"'] & \\
\circ \arrow[rr, "{K^{BS}\,(A,k)}"] & & \circ
\end{tikzcd}$$

A's resulting property enhances what she could deduce from KD_2^3 with the certainty that the opaque submessage M she receives is $K^{BS}\,(A,k)$:

$$\begin{array}{l} A : \mathsf{uncomp}(K^{AS},[A,S]) \ \wedge\ \mathsf{honest}\ S \wedge \\ \quad \langle A,B\rangle_A < (K^{AS}\,(B,k,M))_A < \langle M\rangle_A \\ \Rightarrow \left[\begin{array}{c} \langle A,B\rangle_A \\ (A,B)_S < (\nu\,k)_S < \langle K^{AS}\,(B,k,K^{BS}\,(A,k))\rangle_{S<} \end{array}\right] < \\ \quad < (K^{AS}\,(B,k,\boxed{K^{BS}\,(A,k)}))_A \ =\ (K^{AS}\,(B,k,\boxed{M}))_A \end{array}$$

$=$

At first sight, B's view does not significantly differ from what he could infer in KD_2^3:

$$\begin{array}{ll} B : \mathsf{uncomp}(K^{BS},[B,S]) \ \wedge\ \mathsf{honest}\ S \wedge & \\ & (K^{BS}\,(A,k))_B \\ \Rightarrow & (A,B)_S < (\nu\,k)_S < \langle K^{AS}(B,k,K^{BS}(A,k))\rangle_{S<} < \\ & < \langle K^{BS}\,(A,k)\rangle_{X<} < (K^{BS}\,(A,k))_B \end{array}$$

Assuming S honest and K^{BS} uncompromised, B can deduce that S did its part in the protocol, and that some principal X forwarded $K^{BS}\,(A,k)$ to him. Under the additional assumption that K^{AS} is not compromised either, B can infer that it is A who forwarded this message to him. In particular, this tells B that A knows k.

$$\begin{array}{ll} B : \mathsf{uncomp}(K^{BS},[B,S]) \ \wedge\ \mathsf{honest}\ S \wedge & \\ & \mathsf{uncomp}(K^{AS},[A,S]) \ \wedge\ (K^{BS}\,(A,k))_B \\ \Rightarrow & (A,B)_S < (\nu\,k)_S < \langle K^{AS}\,(B,k,K^{BS}\,(A,k))\rangle_{S<} < \\ & < \langle K^{BS}\,(A,k)\rangle_{A<} < (K^{BS}\,(A,k))_B \end{array}$$

Note that the assumption of $\mathsf{uncomp}(K^{AS},[A,S])$ would be irrelevant in any of B's previous inferences. Note also that the assumption that K^{AS} is uncompromised does not mean that A is bound to be honest: she could indeed deviate substantially from the protocol, passing information (but not K^{AS}) to arbitrary parties, but she certainly has decrypted S's message and certainly sent out $K^{BS}\,(A,k)$ (although not necessarily to B).

While most academic and industrial key distribution protocols based on shared keys are derived from either KD_2^3 or KD_2^4, these fragments lack two important guarantees: *recency* and *key confirmation*. Both KD_2^3 and KD_2^4 give the clients A and B assurance that the key k has been generated by the server for their exclusive communication needs, but they provide no verifiable guarantee that k was generated recently: an old k is more likely to have been compromised than one produced within a short time frame. None of the properties in this section binds the generation of k by any event controlled by the client receiving it. Key confirmation is about a client having some reason to believe that his counterpart has knowledge of k as well: only KD_2^4's B is able to gather this type of evidence (under assumptions). In the next sections, we will follow the development of two known families of protocols and observe how they address these issues.

4 Derivations of NSSK

This section extends the results we just obtained in the direction of the Needham-Schroeder shared-key protocol (NSSK) [12]. In Section 4.1, we describe how a challenge-response exchange is used to guarantee the recency of the key, but also point out how a partial application of this technique leads to Denning and Sacco's classical attack on NSSK [6]. We then show how Needham and Schroeder's subsequent fix to the original NSSK [13] completes the application of nonce-based recency in Section 4.2. Finally, we address key confirmation in Section 4.3.

4.1 Guaranteeing Recency with Nonces

The core key distribution protocols in Section 3 do not guarantee to the clients that the server has generated the key recently. Indeed, none of the derived client formulas bounds the actions of an honest server: the key could have been produced at an arbitrary moment in the past, and now replayed. A client can assure key recency by bracketing its generation between two events whose occurrence it can guarantee. One approach to doing so is using the challenge-response mechanism: the client issues a challenge at the time she sends the key distribution request to the server. The server cryptographically binds the response to the challenge to the key distribution request. We dedicate this section to examining one concrete realization of this idea, adopted in NSSK and other protocols. A different approach, using time-stamps, will be examined in Section 5 when analyzing the Kerberos family.

We use the specific instance of the (cr) axiom shown in Section 2.2, which sends the challenge in the clear (the challenge function is the identity) and returns the response encrypted with an uncompromised shared key. Pictorially:

$$\begin{tikzcd} A & S \\ \circ \arrow[d, "\nu\,n"'] & \\ \circ \arrow[r, "n"] & \circ \arrow[d] \\ \circ & \circ \arrow[l, "K^{AS}\,n"'] \end{tikzcd}$$

This challenge-response is composed with protocol KD_2^4 by means of the *merging transformation* exhaustively studied in [11]. It embeds these messages within the exchange between A and S in this protocol. The overall process is de-

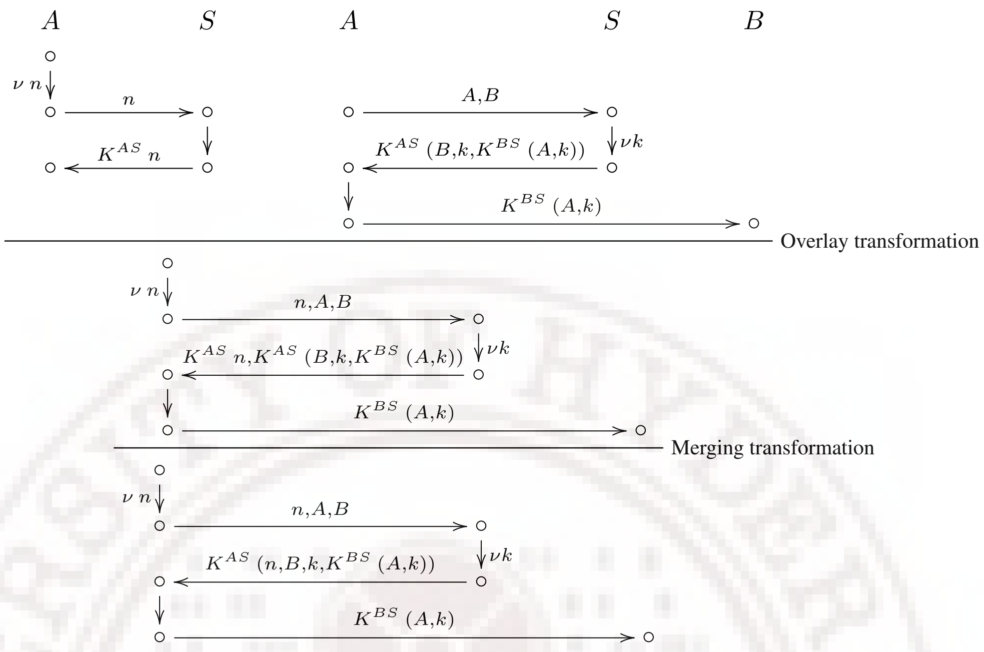
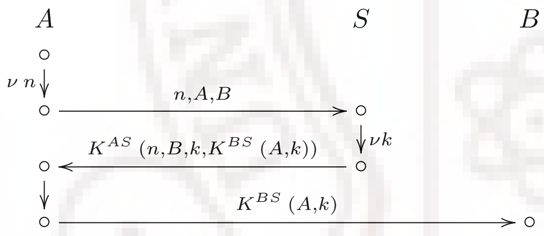


Figure 2. Derivation of NSSK_0

picted in Figure 2. The following diagram reports its effect:



The resulting protocol, NSSK_0, includes the first three steps of NSSK (the addition of key-confirmation will complete it in Section 4.3). B's role does not change at all from KD_2^4; the server's is modified to return the nonce n; most changes occur in A's role.

It is interesting to compare how the properties derivable to A and B change from what we obtained for KD_2^4. Because A created the nonce n fresh and it is returned cryptographically authenticated together with the key k, A can be certain that the server has generated k *after* her request. Thus, NSSK ensures the recency of the key to A.

$$\begin{aligned}
A : \ & \mathsf{uncomp}(K^{AS},[A,S]) \ \wedge \ \mathsf{honest}\ S \ \wedge \\
& (\nu\, n)_A < \langle n, A, B\rangle_A \qquad < (K^{AS}(n,B,k,M))_A \\
\Rightarrow \ & (\nu\, n)_A < \langle n, A, B\rangle_A < (n,A,B)_S < \\
& < (\nu\, k)_S < \langle K^{AS}(n,B,k,K^{BS}(A,k))\rangle_{S<} < \\
& < \qquad\qquad (K^{AS}(n,B,k,K^{BS}(A,k))_A
\end{aligned}$$

The guarantees derivable to B are however much the same as in KD_2^4: B gets to deduce that some nonce n has been exchanged from S's honesty. However, no event controlled by B necessarily precedes the generation of k:

$$\begin{aligned}
B : \ & \mathsf{uncomp}(K^{BS},[B,S]) \ \wedge \ \mathsf{honest}\ S \ \wedge \\
& \mathsf{uncomp}(K^{AS},[A,S]) \ \wedge \ (K^{BS}\ (A,k))_B \\
\Rightarrow \ & (n,A,B)_S < (\nu\, k)_S < \langle K^{AS}(n,B,k,K^{BS}(A,k))\rangle_{S<} < \\
& < \langle K^{BS}(A,k)\rangle_{A<} < (K^{BS}(A,k) : X \to B)_B
\end{aligned}$$

Therefore, NSSK does not ensures the recency of the key to B. This is the essence of Denning and Sacco's attack on NSSK [6].

4.2 NSSK-fix

A few years after Denning and Sacco pointed out the absence of recency guarantees for the responder [6], Needham and Schroeder came forth with a "fix" for their original protocol [13]. This adjustment inserts an additional challenge response between B and the server.

B's challenge differs from A's in order to avoid confusion. B generates a nonce n_B (for symmetry we rename A's nonce n_A), sends $K^{BS}(A, n_B)$ and expects it back as $K^{BS}\, n_B$. One can then verify that the properties of this exchange satisfy the challenge-response schema.

Some preliminary work is needed in order to compose NSSK_0 with this exchange. We first need to apply two instances of the relay transformation to the challenge-response in order to put it in the right "shape" for the merging transformation. Finally, we apply the discharging trans-

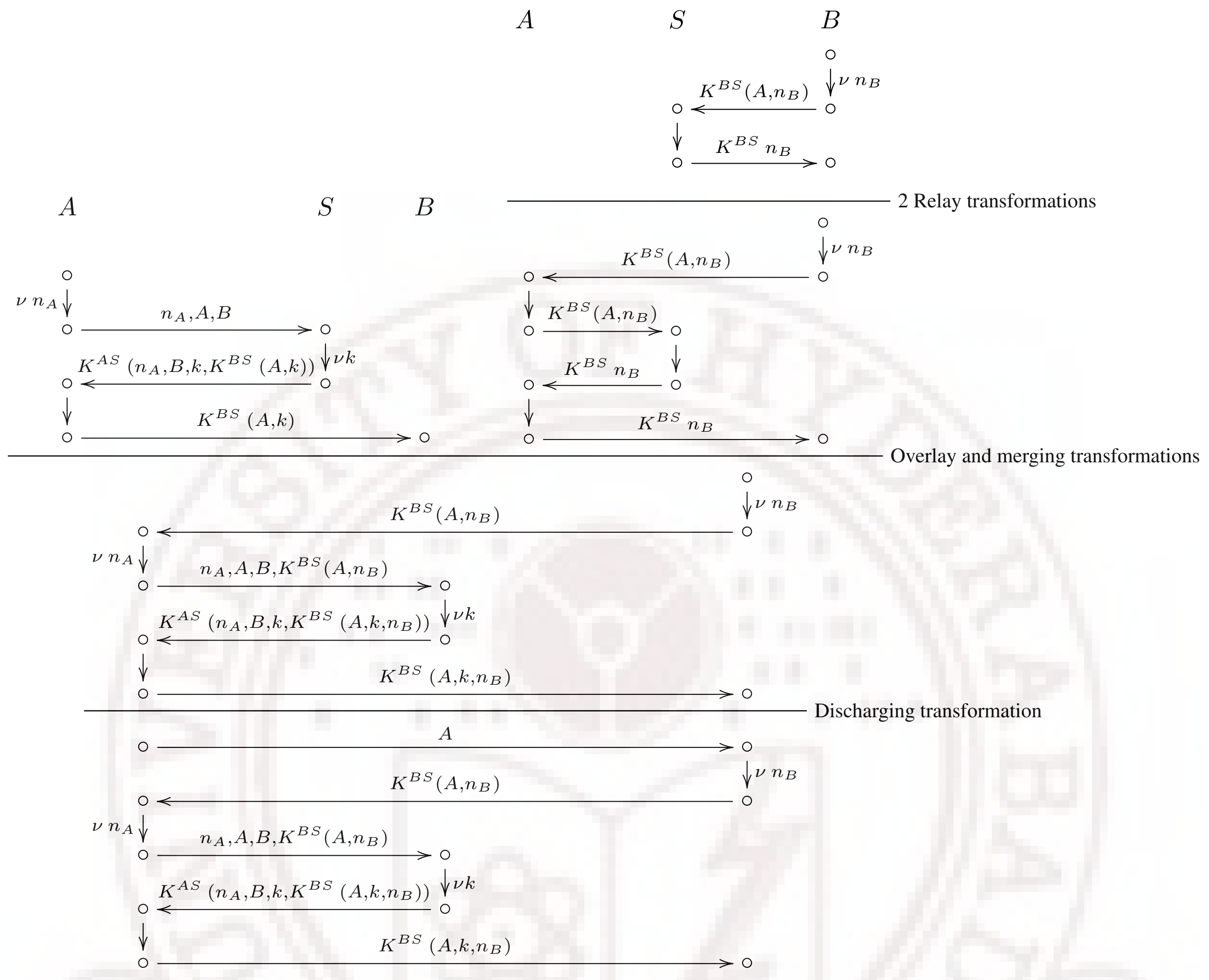


Figure 3. Derivation of $\mathsf{NSSKfix}_0$

formation to maintain A as the initiator of the resulting protocol. This is summarized in Figure 3. We call this protocol $\mathsf{NSSKfix}_0$. Its expected run is as follows:

$$\begin{tikzcd}
A & S & B \\
\circ \arrow[rr, "A"] & & \circ \arrow[d, "\nu\, n_B"] \\
\circ \arrow[d, "\nu\, n_A"'] & & \circ \arrow[ll, "K^{BS}(A,n_B)"'] \\
\circ \arrow[r, "{n_A,A,B,K^{BS}(A,n_B)}"] & \circ \arrow[d, "\nu k"] & \\
\circ \arrow[d] & \circ \arrow[l, "{K^{AS}\,(n_A,B,k,K^{BS}\,(A,k,n_B))}"'] & \\
\circ \arrow[rr, "{K^{BS}\,(A,k,n_B)}"] & & \circ
\end{tikzcd}$$

This protocol differs from NSSK-fix only by the absence of the final key-confirmation steps. They will be added in Section 4.3. Like many other authors, we cannot avoid noting the complexity of this protocol, compared to NSSK or Denning-Sacco.

The lengthy formula characterizing the runs compatible with A's observations does not substantially change the properties available to this principal: if S is honest and K^{AS} is uncompromised, she can still deduce that S has generated k and that he has done so recently.

The interesting changes occur from B's perspective. As in A's case in NSSK_0, B's nonce is cryptographically bound to the key k he receives by protocol's end. Since an honest server will construct this key only after retrieving this nonce from B's encrypted message, the generation of the key is sandwiched between two events under B's control, hence ensuring its recency. The rest of this property allows him to draw similar conclusions as in NSSK_0, namely that S produced the key, forwarded it to A who learned it and forwarded it to B. This is summarized in the following property.

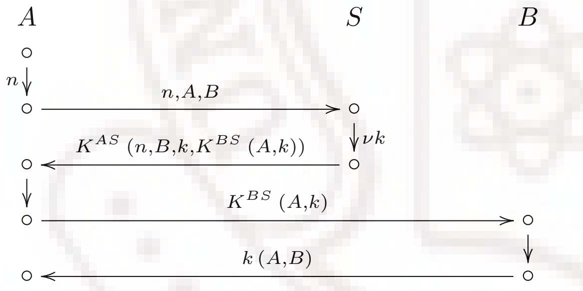
$$\begin{aligned}
&B : \mathsf{uncomp}(K^{BS},[B,S]) \;\wedge\; \mathsf{honest}\ S \;\wedge\\
&\quad \mathsf{uncomp}(K^{AS},[A,S]) \;\wedge\\
&(A)_B < (\nu\, n_B)_B < \langle K^{BS}(A,n_B)\rangle_B < (K^{BS}(A,k,n_B))_B\\
&\Rightarrow\; (A)_B < (\nu\, n_B)_B < \langle K^{BS}(A,n_B)\rangle_B <\\
&\quad < (n_A, A, B, K^{BS}\,(A,n_B))_S < (\nu\, k)_S <\\
&\quad < \langle K^{AS}\,(n_A, B, k, K^{BS}\,(A,k,n_B))\rangle_{S<} <\\
&\quad < \langle K^{BS}\,(A,k,n_B)\rangle_{A<} < (K^{BS}\,(A,k,n_B))_B
\end{aligned}$$

As in NSSK_0, dropping the assumption that K^{AS} is uncompromised implies that B does not know who has originated the message $K^{BS}\,(A,k,n_B)$ and that he cannot be certain that A knows k.

4.3 Key Confirmation

The previous two sections have shown how to extend the core key distribution protocol KD_2^4 with the recency guarantees of NSSK(-fix). The remaining issue, addressed in this section, is ensuring to both recipients that their counterpart also knows the new shared key. So far, only B has this guarantee.

The simplest way to achieve this is by having B send A a pre-agreed message m (e.g., (A,B)) encrypted with k. Post-composing NSSK_0 with this transmission yields the protocol NSSK_1, which has the following expected run:



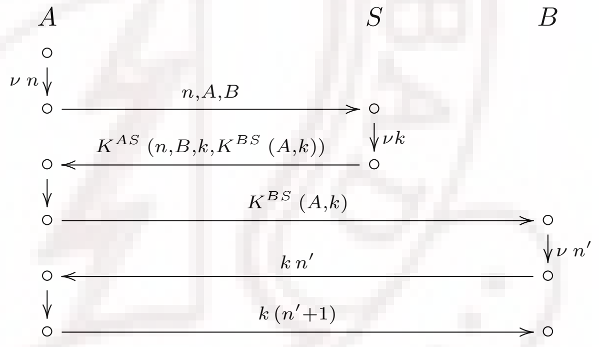
A's observations lead her to conclude:

$$\begin{aligned}
&A : \mathsf{uncomp}(K^{AS},[A,S]) \;\wedge\; \mathsf{honest}\ S \;\wedge\\
&\boxed{\mathsf{uncomp}(k,[A,B,S])} \;\wedge\; (\nu\, n)_A < \langle n,A,B\rangle_A <\\
&\quad < (K^{AS}(n,B,k,M))_A < \boxed{\langle M\rangle_A < (k(A,B))_A}\\
&\Rightarrow\; (\nu\, n)_A < \langle n,A,B\rangle_A \; < (n,A,B)_S < (\nu\, k)_S <\\
&\quad <\; \langle K^{AS}(n,B,k,K^{BS}\,(A,k))\rangle_{S<} <\\
&\quad <\; (K^{AS}(n,B,k,K^{BS}(A,k))_A <\\
&\quad <\; \boxed{\begin{array}{l}\langle K^{BS}(A,k)\rangle_A < (K^{BS}(A,k))_B <\\ < \langle k(A,B)\rangle_{B<} < (k(A,B))_A\end{array}}
\end{aligned}$$

We have highlighted the additions with respect to NSSK_0 (see Section 4.1). We had omitted the then trailing $\langle M\rangle_A$ and $\langle K^{BS}\,(A,k)\rangle_A$ since they did not add substantial information. Now they clearly do, as they allow A to infer that B has received this message and originated $k\,(A,B)$.

The last addition, $\mathsf{uncomp}(k,[A,B,S])$, deserves some discussion. Clearly, we need to know that k is uncompromised to infer anything useful involving it. However, most formal systems would *derive* this fact rather than *assume* it. This may be where the strict separation between authentication and secrecy is most evident in this work. Recall that our logical system is just powerful enough to reason about the order of actions, the structure underlying authentication. In particular it does not embed the logical principles to derive that k must indeed be secret. The assumption $\mathsf{uncomp}(k,[A,B,S])$ is an interface to a secrecy logic.

Applying the above extension to $\mathsf{NSSKfix}_0$ yields $\mathsf{NSSKfix}_1$. This protocol has then the typical properties of a key distribution protocol: both clients receive assurance that the key has been generated by the expected server, that this key is controllably recent, and that they both know the key. However, the actual NSSK-fix is different: B encrypts a new nonce with k and sends it to A, and expects this same nonce back from A, transformed in a predictable way. We will now discuss what additional properties are achieved by doing so. For the sake of succinctness, we operate on NSSK_1, which differs from the original NSSK in precisely the same way as $\mathsf{NSSKfix}_1$ is different from NSSK-fix. Here is the expected run of NSSK:



First, notice that having A send something encrypted with k back to B does not produce any new knowledge (besides the obvious, i.e., that a new message has been transmitted). From the point of view of B, the last two messages of NSSK implement a challenge-response exchange: B generates the nonce n', sends it to A encrypted (with k), and expects it back from her transformed. B thus ascertains that A in alive at this particular point of the protocol. Note that B could repeat this same exchange an arbitrary number of times (each with a new nonce) and obtain the same guarantee: that A was recently alive. De facto, this implements a crude single-authentication, repeated-request client-server mechanism, with the initiator acting as the server and the responder as the client.

In summary, our analysis shows that NSSK-fix achieves key distribution with recency guarantees and key confirmation for both parties. NSSK provides recency assurance only to the initiator. Our work also shows that the same guarantees are also supported by simpler protocols that drop the last message and rely on any pre-arranged message instead of the final nonce.

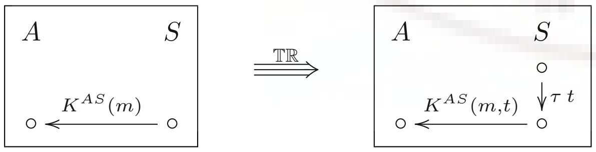
5 Derivations of Kerberos

Kerberos [14, 15] is a complex and versatile protocol that has been the subject of intense scrutiny over the years. In this section, we will apply the methods outlined above to derive the core authentication functionalities of versions 4 and 5 of this protocol. We concentrate on the basic key distribution exchange of which each version contains two instances. As a preparatory step, we formalize the use of timestamps for authentication and derive the Denning-Sacco protocol, a core component of Kerberos 4.

5.1 Guaranteeing Recency with Timestamps

Timestamps have a number of applications in cryptographic protocols. In this section, we examine and formalize their use for the purpose of guaranteeing the recency of an already authenticated message. Consider a principal A receiving a message $K^{AS}\ m$ from an honest agent S: if the key is uncompromised, A can only deduce that S originated this message in the (possibly distant) past; if however S includes a timestamp t within the encryption and sends $K^{AS}(m,t)$, A can assess the age of the message and reject it if it falls outside of her window of validity. This assessment takes into considerations clock skews between hosts, typical network delays, etc.

We formalize this intuition as a new variant of the binding refinement [5] used in Section 3 in which the bound term is a timestamp. We call it the *timestamping refinement* and denote it $\mathbb{TR}$. It transform the exchange on the left below into the exchange on the right:



This refinement allows upgrading the logical guarantees that each principal can deduce. Given the particular format of this transformation (S does not receive a message back), we concentrate on the formulas derivable by A. Schematically:

$$\begin{array}{ll} A: & \mathsf{uncomp}(K^{AS},[A,S]) \wedge (\!(K^{AS}\ m)\!)_A \\ \Rightarrow & \langle\!\langle K^{AS}\ m\rangle\!\rangle_{S<} < (\!(K^{AS}\ m)\!)_A \end{array}$$

$$\Big\Downarrow \mathbb{TR}$$

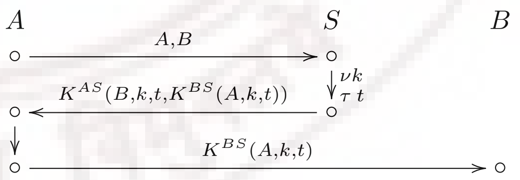
$$\begin{array}{l} A: \mathsf{uncomp}(K^{AS},[A,S]) \wedge \mathsf{honest}\ S \wedge (\!(K^{AS}(m,t))\!)_A \\ \Rightarrow (\underline{\tau}\ t)_A < (\tau\ t)_S < \langle\!\langle K^{AS}(m,t)\rangle\!\rangle_{S<} < (\!(K^{AS}(m,t))\!)_A \end{array}$$

The top formula describes how A can extend her knowledge after receiving $K^{AS}\ m$ whenever the original protocol guarantees the authenticity of m: note that, as long as K^{AS} is not compromised, S is not required to be honest. The bottom lines show the upgraded formula. Recall that $(\underline{\tau}\ t)$ represents the earliest point in A's local time where she will accept the time t as valid. It is now important that S is believed to be honest: without this, S could guess an appropriate value for t rather than looking it up on its clock.

We obtain this formula by homomorphically replacing $K^{AS}\ m$ with $K^{AS}(m,t)$ in the derivation of the top formula. The atom $(\tau\ t)_S$ comes from the upgraded honesty axiom and is justified by axiom (ts).

5.2 The Denning-Sacco Protocol

The Denning-Sacco protocol [6] applies the timestamping refinement to the basic key distribution protocol with nested encryption KD_2^4 where the authenticated message (m above) is (k, X), where k is the newly generated key and X is either A or B. S applies this refinement twice, adding the same timestamp next to each key distribution submessage. As a consequence, by the completion of the protocol, each principal has the certainty that S has generated k recently. As in KD_2^4, because of the nested encryption, B additionally knows that A has seen k (but A cannot be certain that B ever receives k). We have shown the resulting protocol in Section 2. Its expected run is as follows:



From the sole observation of her actions and the honesty of the server, A can reconstruct the whole protocol, save for B's reception of her last message:

$$\begin{array}{l} A: \mathsf{uncomp}(K^{AS},[A,S]) \ \wedge\ \mathsf{honest}\ S\ \wedge \\ \quad \langle A,B\rangle_A \qquad\qquad\qquad < (K^{AS}(B,k,t,M))_A \\ \Rightarrow \begin{bmatrix} \langle A,B\rangle_A < (A,B)_S \\ (\underline{\tau}\ t)_A \end{bmatrix} < \begin{bmatrix} (\nu\ k)_S \\ (\tau\ t)_S \end{bmatrix} < \\ \quad < \ \langle K^{AS}(B,k,t,K^{BS}(A,k,t))\rangle_{S<} \\ \qquad\qquad < \ (K^{AS}(B,k,t,K^{BS}(A,k,t)))_A \end{array}$$

We have elided A's final send action as it does not contribute added knowledge. Note that S's generation of k is now bounded by $\underline{\tau}\, t$, which is under the control of A.

B's conclusions merge the recency assurance provided by timestamps with what he could infer by means of KD_2^4, i.e., that S has generated k and that A has seen it in order to forward the message he receives:

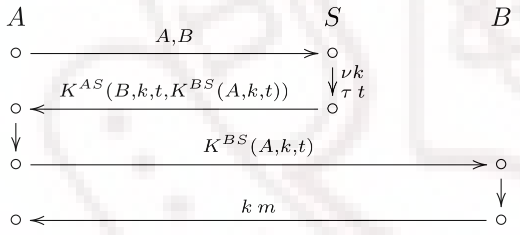
$$\begin{aligned} B : \ & \mathsf{honest}\ S \ \wedge\ \mathsf{uncomp}(K^{BS}, [B,S]) \ \wedge \\ & \wedge\ \mathsf{uncomp}(K^{AS}, [A,S]) \ \wedge\ (K^{BS}(A,k,t))_B \\ \Rightarrow\ & \begin{bmatrix} (A,B)_S \\ (\underline{\tau}\, t)_B \end{bmatrix} < \begin{bmatrix} (\nu\, k)_S \\ (\tau\, t)_S \end{bmatrix} < \\ & < \langle K^{AS}(B,k,t,K^{BS}(A,k,t))\rangle_{S<} < \\ & < \ \langle K^{BS}(A,k,t)\rangle_{A<} < \ (K^{BS}(A,k,t))_B \end{aligned}$$

Denning and Sacco prominently pointed out in their original paper [6] that this protocol provides full recency guarantees with a minimum number of messages.

5.3 Kerberos 4

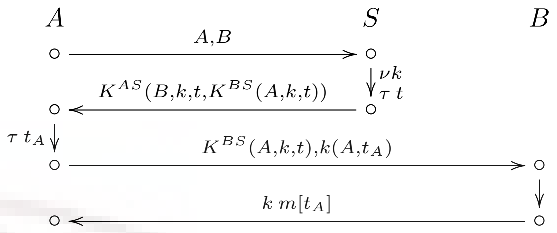
The core authentication functionalities of Kerberos 4 [14] are obtained by simply extending the Denning-Sacco protocol by means of a key confirmation exchange similar to the way we obtained NSSK(-fix) in Section 4.3.

Adding key confirmation We extend DS by having B send A some (recognizable) message m encrypted with k. The resulting run is as follows:



The formula characterizing the runs compatible with each principal's observations is extended as in Section 4.3 [4].

Adding repeated authentication Kerberos was designed as a *repeated* authentication protocol: each time A presents the *ticket* $K^{BS}(A,k,t)$, B will provide some predetermined service (up to an end-date that we can abstractly think of as a function of t). The protocol we just derived is clearly inadequate for this purpose as anybody can replay the ticket $K^{BS}(A,k,t)$. B needs to authenticate that a subsequent request comes from A, and assess that it was made recently enough. Kerberos 4 realizes these two goals by having A generate a timestamp t_A just prior to issuing a new request, and embedding into it an *authenticator* $k\,(A,t_A)$ (any message mentioning t_A and encrypted with k would do). The intended run of the resulting protocol is as follows:



where the last message is made dependent on t_A (although Kerberos does not always enforce this).

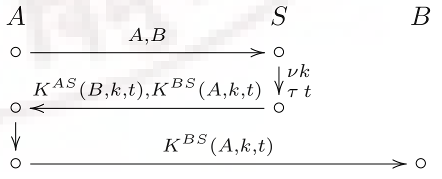
Observe that, differently from NSSK(-fix), it is the initiator of the protocol (the client, A) that requests the service provided by the responder (B). Indeed, A generates the timestamp t_A that is included in the authenticator.

Kerberos 4 [14] extends this core protocol with numerous fields primarily meant to negotiate parameters of the resulting authentication: added timestamps, options and flags, access control information, etc. For maximum flexibility, Kerberos chains two instances of the core protocol, by which a client (A) first obtains a master ticket (TGT) which simplifies the issuance of tickets for individual services.

5.4 Kerberos 5

As far as authentication is concerned, Kerberos 5, the most recent version of this protocol [14, 15], differs from Kerberos 4 only by the form of the basic key distribution mechanism it relies on: while version 4 was built up from the nested variant KD_2^4 , Kerberos 5 starts with the concatenated variant KD_2^3. Given this different starting point, the core protocol is however derived by applying the exact same steps as for Kerberos 4. It is interesting to examine them as the conclusions available to the various principals are not the same throughout.

The concatenated variant of the Denning-Sacco protocol has the following expected run:



The knowledge derivable by A is similar to the Denning-Sacco protocol, except that she can never be certain that the encrypted component she receives corresponds to what S has sent. More interesting is the knowledge inferable by B: differently from the Denning-Sacco protocol, B cannot reach any conclusion on whether A ever saw the key k:

indeed, the assumption $\mathsf{uncomp}(K^{AS}, [A, S])$ becomes irrelevant. B knows that the server sent the appropriate messages and that some principal X forwarded the correct component to him. This makes B's knowledge very similar to A's.

With both A and B unaware of whether its counterpart has seen k, each party needs to inform the other of its knowledge of k. We rely on the device already used in Kerberos 4 to accomplish this: A will concatenate the component $(k\,A)$ as she forwards $K^{BS}(A, k, t)$ to B. As in version 4, B will confirm k with a response $k\ m$ for some recognizable m. We obtain the following exchange:

$$\begin{tikzcd}
A & S & B \\
\circ \arrow[r, "A{,}B"] & \circ \arrow[d, "\nu k \atop \tau t"] & \\
\circ \arrow[d] & \circ \arrow[l, "K^{AS}(B{,}k{,}t){,}K^{BS}(A{,}k{,}t)"'] & \\
\circ \arrow[rr, "K^{BS}(A{,}k{,}t){,}k\ A"] & & \circ \arrow[d] \\
\circ & & \circ \arrow[ll, "k\ m"']
\end{tikzcd}$$

This protocol fragment is extended to allow repeated authentication using k exactly as for Kerberos 4: A generates a timestamp t_A and includes it in her authenticator; B optionally returns t_A in the last message.

This is the authentication core of Kerberos 5. As in its predecessor, two instances of this fragment are chained together, and numerous fields add a great deal of flexibility [14, 15]. It should be noted that, in Kerberos 5, the timestamp-based recency assessment (using t) is supplemented with a nonce-based guarantee by which A sends S a nonce n with her initial request and expects it back within $K^{AS}(B, k, t)$.

6 Conclusions

We have proposed a simple logic of partial order to analyze the authentication properties of security protocols. It encapsulates necessary secrecy guarantees as assumptions to be proved in a different formalism. This clean separation was motivated by the observation that secrecy and authentication properties rely on very different proof methods, often intertwined in complex analyses. We use this logic to drive the derivational approach in organizing a large class of authentication protocols into a taxonomy cataloged by the authentication properties of their components and how they are combined.

In a natural continuation of this work, we are developing a logic of pure secrecy that encapsulates authentication requirements as assumptions. The authentication logic is being incorporated into the Protocol Derivation Assistant [1]. We are also currently investigating the automation of the process by which a principal's observation is completed into the set of compatible runs. Efforts in this direction touch the question of the decidability of verifying authentication (with encapsulated secrecy assumptions).

References

[1] M. Anlauff and D. Pavlovic. The Protocol Derivation Assistant (version 1.8.21). Available electronically at `http://www.kestrel.edu/software/pda`, Mar. 2005.

[2] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, 1990.

[3] F. Butler, I. Cervesato, A. D. Jaggard, and A. Scedrov. A Formal Analysis of Some Properties of Kerberos 5 using MSR. In *Proc. CSFW-02*, pages 175–190. IEEE Computer Society Press, 2002.

[4] I. Cervesato, C. Meadows, and D. Pavlovic. Deriving key distribution protocols and their security properties. 2005.

[5] A. Datta, A. Derek, J. Mitchell, and D. Pavlovic. A derivation system and compositional logic for security protocols. *Journal of Computer Security*, 2004. To appear.

[6] D. E. Denning and G. M. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536, 1981.

[7] W. Diffie, P. C. van Oorschot, and M. J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes, and Cryptography*, 2:107–125, 1992.

[8] N. Durgin, J. Mitchell, and D. Pavlovic. A compositional logic for proving security properties of protocols. *Journal of Computer Security*, 11(4):667–721, 2003.

[9] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. *Software - Concepts and Tools*, 17:93–102, 1996.

[10] G. Lowe. A herarchy of authentication specifications. In *Proc. 10th IEEE Computer Security Foundations Workshop*, pages 31–43. IEEE Computer Society Press, 1997.

[11] C. Meadows and D. Pavlovic. Deriving, attacking and defending the GDOI protocol. In *Proc. ESORICS 2004*, pages 33–53. Springer-Verlag LNCS 3193, 2004.

[12] R. M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[13] R. M. Needham and M. D. Schroeder. Authentication revisited. *ACM Operating Systems Review*, 21(1):7, 1987.

[14] B. C. Neuman and T. Ts'o. Kerberos: An Authentication Service for Computer Networks. *IEEE Communications*, 32(9):33–38, 1994.

[15] C. Neuman, J. Kohl, T. Ts'o, T. Yu, S. Hartman, and K. Raeburn. The Kerberos Network Authentication Service (V5), September 7 2004. Internet draft, expires 7 March 2005.

A logic-based verification framework for authentication protocols

Shahabuddin Muhammad*, Zeeshan Furqan and Ratan K. Guha

School of Electrical Engineering and Computer Science,
University of Central Florida, Orlando, FL 32816, USA
Fax: +1 4078235419 E-mail: muhammad@cs.ucf.edu
E-mail: zfurqan@cs.ucf.edu E-mail: guha@cs.ucf.edu
*Corresponding author

Abstract: Designing authentication protocols is an error-prone process. In this paper, we develop a deductive style proof-based framework to verify authentication protocols. The proposed framework clearly represents authentication protocols and concisely proves their security properties. We utilise Distributed Temporal Protocol Logic (DTPL) to capture temporal aspects of distributed events. We formalise essential features for achieving authentication. We also extend the notion of source association for public-key protocols. The resulting framework demonstrates the ease with which a protocol is analysed. It also demonstrates DTPL's suitability to be used as a meta-level tool to benefit from different formalisms.

Keywords: security protocols; authentication; distributed temporal protocol logic; DTPL; SVO logic.

Reference to this paper should be made as follows: Muhammad, S., Furqan, Z. and Guha, R.K. (2007) 'A logic-based verification framework for authentication protocols', *Int. J. Internet Technology and Secured Transactions*, Vol. 1, Nos. 1/2, pp.49–80.

Biographical notes: Shahabuddin Muhammad is a PhD candidate in the School of Electrical Engineering and Department of Computer Science in the College of Engineering and Computer Science at the University of Central Florida. He is expected to graduate in August 2007. His area of research includes security protocols, formal verification, and distributed systems.

Zeeshan Furqan is also a PhD candidate in the School of Electrical Engineering and Department of Computer Science in the College of Engineering and Computer Science at the University of Central Florida. He is also expected to graduate in August 2007. His area of research includes security protocols, 802.11i, and protocol design.

Ratan K. Guha received the BSc Degree with Honours in Mathematics and the MSc Degree in Applied Mathematics from Calcutta University and received the PhD Degree in Computer Science from the University of Texas at Austin. He is currently a Professor of Computer Science at the University of Central Florida, Orlando. His research interests include distributed systems, computer networks, security protocols, modelling and simulation, and computer graphics. He is a member of the ACM, IEEE, and SCS and served as a member of the Board of Directors of SCS from 2004 to 2006.

1 Introduction

Distributed systems have been in use for commercial purposes as well as in sensitive domains, such as defence, for many years. E-commerce has also become a widely used successful tool to promote businesses on the internet. In scenarios where critical information is being widely communicated across distributed networks, security becomes an extremely important issue. Before initiating a financial transaction, one needs to make sure that its sensitive data will be securely transmitted to the intended recipient. Several types of encryption schemes have been developed to ensure secure transmission of data. However, history has shown that even assuming ideal cryptography,[1] one can not assure that its secrets will safely reach to the intended destination. The reason for this lack of security is not the underlying cryptography, but the security holes in the cryptographic protocols themselves.

A security protocol (or cryptographic protocol) is a sequence of messages between two or more parties in which encryption is used to provide authentication or to distribute cryptographic keys for new conversations (Needham and Schroeder, 1978). The network is assumed to be hostile as it contains intruders with the capabilities to encrypt, decrypt, copy, forward, delete, and so forth. Considering an active intruder with such powerful capabilities, it becomes extremely difficult to guarantee proper working of a security protocol. Several examples show how carefully designed protocols were later found out to have security breaches (Muhammad et al., 2006). This situation led the researchers to formalise the verification of security protocols.

Logic-based verification is one of the widely used formal verification techniques in the domain of security protocols. This is due to both, the simplicity of the logic-based methods and the conciseness of the proof they generate (Coffey et al., 2003). The logic of belief of Burrows et al. (1990), known as BAN logic, provided the initial impetus in applying modal logic in a proof-based environment. Analysing a protocol in BAN begins by first formalising the message exchanges of the protocol in the language of its logic. Then all the initial assumptions of the protocol and assertions about each message exchange are written. Finally, BAN rules are applied on the assumptions and the assertions in order to derive the conclusion. Although BAN logic has also been criticised for various reasons explained in the next section, it has been successfully used to discover various attacks on well-known protocols. With the emergence of BAN, several researchers have applied various logic-based techniques for the formal verification of security protocols. BAN logic has also been extended in many ways (Gong et al., 1990; van Oorschot, 1993; Shand and Vuillemin, 1993; Abadi and Tuttle, 1991).

Recently a new logic, called Distributed Temporal Protocol Logic (DTPL), has been proposed in Caleiro et al. (2005a) which provides an object-level tool to model distributed communication. DTPL's distinguishing characteristics is its capability to be used as a metalevel tool for comparative analysis of security protocol models and properties. In this paper, we have developed a BAN-like proof system based on DTPL. We devise a logic-based verification system that involves the notion of derivability (i.e., proofs) in which a formula φ is derivable from a set of formulae Γ using the inference rules. Since DTPL provides an intuitive framework that captures reasoning in a distributed environment, we utilise the existing DTPL and develop it such that it can be used to verify security protocols in a proof-based setting. For this purpose, we take advantage from the work of Syverson and Cervesato (2001) (SVO logic). The reason for using SVO is that it clarifies many concepts in previous logics and unifies four of its predecessors in a sound way. We demonstrate how DTPL in corporates the essential features for protocol verification of SVO in its framework. We also describe how DTPL helps us clarify some concepts, such as freshness, in the existing logic-based techniques. As in other logic-based techniques, we see security protocols to possess essential ingredients to guarantee authentication, such as message freshness, key association, message origination etc. We represent these ingredients in terms of DTPL. Authentication protocols can be broadly categorised based on different types of cryptography they use and on how they achieve authentication goal. In particular, when finding the originator of a received message, we categorise authentication protocols into symmetric-key, asymmetric-key, and challenge-response protocols.

Our verification framework contains the advantages of both, the expressibility of the existing DTPL model as well as the conciseness of a proof-based system. Moreover, due to DTPL's rich interpretation structure of Winskel (1987), analysing a protocol becomes clearer as compared to other logic-based techniques. Since our work is based on the notion of derivability, the verification process does not explicitly model an intruder, thereby obviating the need to apply all combinations of intruder behaviours for protocol analysis. This results in considerable simplicity in the way a proof-based method analyses a protocol. In addition to developing the existing DTPL framework for three types of protocols, we demonstrate the applicability of our work by applying it on a well-known protocol.

We have organised the rest of the paper as follows. Section 2 provides a brief background of the field, specifically the BAN logic and some of its extensions. In Section 3, we summarise the essential features of DTPL. Section 4 describes how the SVO logic can be represented in the DTPL framework. In particular, we describe some the similarities and differences and how to overcome those differences in the language of both logics. Section 4 also demonstrates how SVO rules can be viewed in the DTPL framework. Finally, we present the DTPL translation of SVO's source association axioms and its extension for public-key protocols in that section. We present an example protocol to illustrate the applicability of our work in Section 5. We conclude in Section 6.

2 Background

Over the past two decades, formal verification has been rigorously applied in the domain of security protocols. Researchers have developed numerous formal techniques and tools and successfully applied them to either verify a security property or to catch a flaw in a security protocol. Dolev and Yao (1983) contributed the early work in the application of formal methods to verify cryptographic protocols. Since then, legions of new methods based on formal verification began to occupy the attention of researchers. We do not intend to provide an exhaustive survey of the field. The interested readers should see Meadows (2003), for a good survey encompassing the field of formal verification of security protocols. Generally speaking, most of the proposed schemes can be broadly categorised into logic-based techniques, process algebra, theorem proving, and model checking approaches (Fidge, 2001). The approaches in Burrows et al. (1990), Gong et al. (1990), Abadi and Tuttle (1991), Oorschot (1993) and Syverson and Oorschot (1994) use modal logic in their frameworks, and Bodei et al. (2002) and Ryan et al. (2000) are the examples of techniques based on process algebra. Paulson's Isabelle (Paulson, 1998) is an example of a theorem prover, whereas the tools proposed in Basin et al. (2005), Millen and Shmatikov (2001), Song et al. (2001), Kemmerer (1989), Longley and Rigby (1992), Meadows (1992), Millen et al. (1987) and Marrero et al. (1997) can be classified as model checkers. All of these techniques, with their own merits and demerits, have been successfully used to find flaws in many protocols.

Recently a new logic, based on the extension of the distributed temporal logic DTL of Ehrich and Caleiro (2000), has been proposed for security protocols (Caleiro et al., 2005a). The DTPL provides an object-level tool which can model protocol-independent distributed communication. DTPL uses a model-checking approach in which a model checker explores all the possible states in a model in search for a state where a given specification (property) does not hold. In case of a successful termination, the model checker generates a counterexample. In the domain of security protocols, a counterexample represents an attack on the protocol. Hitherto, several model-checkers have been designed and successfully applied to discover various attacks on security protocols. For example, Lowe's FDR was used to identify the famous flaw in the Needham-Schroeder public-key protocol (Lowe, 1996). However, the most severe problem in model-checking approaches is that of the state explosion. Many techniques have been suggested and applied in various model checkers in order to reduce the state-space explosion problem. In the domain of security protocols, partial-order and symmetry reductions have been used, for example, in Clarke et al. (2000). In fact, DTPL is also mentioned as being used in this direction in Caleiro et al. (2005a).

As opposed to the model-checking approach, a proof-based system focuses on providing a concise proof of the desired goals. The seminal work of applying logics in a proof-bases environment is the 'logic of belief' by Burrows, Abadi, and Needham, known as BAN logic (Burrows et al., 1990).[2] The way a protocol is analysed in BAN involves a process of deductive reasoning, where the desired protocol goals are deduced by applying a set of axioms and inference rules to the assumptions and messages of the protocol. One of the many reasons for the popularity of BAN logic was its simplicity and conciseness of the proofs it generates. If a protocol is successfully verified, it is considered secure within the scope of the logic. On the other hand, inability to derive the desired goals suggests possible weaknesses which helps redesign the protocol. However, verification logics have also been criticised due to the problems associated with them. For instance, these techniques require one to formalise protocol messages into the language of the logic. In BAN, it is called *idealisation* which involves proper understanding of the protocol as well as clearly identifying the intention of the sender about the messages it sends. This increases the likelihood of different formalisation of the same protocol resulting from several interpretation of protocol messages. Therefore, a formally verified protocol does not guarantee that the protocol is secure in general, but only guarantees that the idealised version of the protocol is secure. Another criticism on BAN logic was the lack of clarity in expressing the protocol goals and the initial assumptions (Coffey et al., 2003). For example, the authors of BAN made some 'dubious' assumptions when they analysed some well-known protocols (Burrows et al., 1990). This required deep understanding of the protocol, in contrast to the simplicity that logic-based methods intended to offer.

In response to these criticisms, several researchers extended the BAN logic in an effort to clarify some misconceptions and to increase its applicability to a wider range of protocols (Gong et al., 1990; Abadi and Tuttle, 1991; Syverson and Oorschot, 1994; Syverson and Cervesato, 2001). Their frameworks not only tried to solve the problems in BAN, but also attempted to maintain its simplicity. The logic of SVO (Syverson and Cervesato, 2001) is such an attempt which unifies four of its predecessors BAN (Burrows et al., 1990), GNY (Gong et al., 1990), AT (Abadi and Tuttle, 1991), and VO (van Oorschot, 1993). All of these logics have been used to generate concise proofs and have successfully identified a number of flaws in many protocols.

Since our work focuses on developing a proof-based method from DTPL, we briefly describe the pioneer work in applying logics in a proof-based setting in this area, that is, BAN logic. We also discuss some of the successors (like GNY, AT, and VO) of BAN in this section. Finally, we describe the logic that we have used in this paper, that is, SVO logic, which is a unification of some of the successors of BAN logic.

2.1 The logic of BAN

BAN (Burrows et al., 1990) is a logic of belief which was the first attempt in applying modal logic to verify security protocols. Since security protocols involve principals sending and receiving messages to each other, each principal holds certain beliefs about these messages. For example, a principal *A* believes that a message *M* is fresh. In BAN notations, it is represented by $A \mid\equiv \#(M)$. Since the notations used in BAN are non-intuitive, we replace those notations by their meaning in plain English as done in Abadi and Tuttle (1991) and Syverson and Cervesato (2001). Therefore, we replace $A \mid\equiv \#(M)$ by *A believes fresh*(*M*). Principals also hold beliefs about communication, such as *A believes B said M, A believes B sees M* and so on. Some of the other expressions of the language of BAN are *A controls M* (*A* is trusted on the values of *M*), $A \overset{k}{\longleftrightarrow} B$ (*k* is a shared key between *A* and *B*), and $\overset{k}{\mapsto} A$ (*k* is a public-key of *A*). BAN analyses a protocol by following a sequence of steps and applying a set of rules. We will explain these steps with the help of an example in this section. Some of the important BAN rules include:

Message Meaning Rule: This rule states that if *A* believes *k* to be a shared-secret between him and *B* and he received a message encrypted by *k*, then he is entitled to believe that *B* said *M.* This rule is used to find out the sender of a received message such that some principals already share some secrets.

$$\frac{A\ believes\ A \overset{k}{\longleftrightarrow} B \qquad A\ received\ \{M\}_k}{A\ believes\ B\ said\ M}.$$

Public-key equivalent of this rule can also be written in the similar way.

Nonce Verification Rule: It states that if *A* believes a message *M* to be fresh and he believes that a principal *B* said that message sometime in the past, then he is entitled to believe that *B* still believes in *M* (because of its freshness). This rule is used to make sure that principals do not become victim of replay attack in which an intruder replays an old message.

$$\frac{A\ believes\ fresh(M) \qquad A\ believes\ B\ said\ M}{A\ believes\ B\ believes\ M}.$$

Jurisdiction Rule: It states that if a principal believes in a message *M* such that he has authority over *M* then *M* is believable. It is mainly used for servers who are responsible for generating keys for other principals.

$$\frac{A\ believes\ B\ controls(M) \quad A\ believes\ B\ believes\ M}{A\ believes\ M}.$$

The above-mentioned rules provide the main machinery in achieving security goals, such as authentication, in BAN logic. To facilitate the goal derivation, BAN also provides some other rules as given below.

A believes the concatenation and concatenates of its believed messages:[3]

$$\frac{A\ believes\ X \quad A\ believes\ Y}{A\ believes\ (X,Y)} \quad \frac{A\ believes\ (X,Y)}{A\ believes\ X}.$$

A believes that the other principals believe in the concatenates of their concatenated messages:

$$\frac{A\ believes\ B\ believes(X,Y)}{A\ believes\ B\ believes\ X}.$$

A holds *B* responsible for saying all the concatenates of a said message:

$$\frac{A\ believes\ B\ said(X,Y)}{A\ believes\ B\ said\ X}.$$

A message is fresh if any of its concatenates is fresh:

$$\frac{A\ believes\ fresh(X)}{A\ believes\ fresh(X,Y)}.$$

Given that a principal holds the corresponding decryption key, contents of an encrypted received message are also considered to be received:

$$\frac{A\ believes\ A \overset{k}{\longleftrightarrow} B \quad A\ received\{M\}_k}{A\ received\ M}.$$

The above rule can also be extended for public-key encryption and public-key signature as follows:

$$\frac{A\ believes(\overset{k}{\mapsto} A) \quad A\ received\{M\}_k}{A\ received\ M} \quad \frac{A\ believes(\overset{k}{\mapsto} B) \quad A\ received\{M\}_{k^{-1}}}{A\ received\ M}.$$

Here, k represents a public key whereas k^{-1} represents a private key in asymmetric cryptography. In symmetric cryptography, $k = k^{-1}$. Concatenates of a concatenated received message are also believed to be received:

$$\frac{A\ received\ (X,Y)}{A\ received\ X}.$$

Given the above rules, we can analyse a protocol in BAN logic using the following steps.

1 Idealise a protocol (explained by the following example).

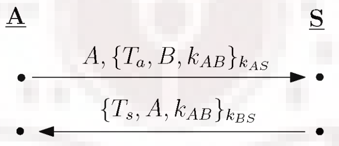
2 State all the initial assumptions of the protocol.

3 For each message transmission in the protocol of the form $A \rightarrow B : M$, write assertion of the form *B received M.*

4 Apply BAN rules on the assumptions and assertions to derive beliefs held by other principals of the protocol.

We use a very simple protocol to demonstrate the application of BAN logic. The "Wide-Mouthed Frog protocol" has been analysed using BAN in Burrows et al. (1990). The protocol comprises of only two steps as shown in Figure 1. In the first message, a principal A sends a session key k_{AB} along with a timestamp T_a to S. After checking the timeliness of the first message, S adds its own timestamp and sends the second message to B.

Figure 1 The wide-mouthed frog protocol


A
S
$A, \{T_a, B, k_{AB}\}_{k_{AS}}$
$\{T_s, A, k_{AB}\}_{k_{BS}}$

Step 1: The idealised protocol is given below.

Message 1. $A \rightarrow S : \{T_a, (A \xleftrightarrow{k_{AB}} B)\}k_{AS}$

Message 2. $S \rightarrow B : \{T_s, A \ believes\,(A \xleftrightarrow{k_{AB}} B)\}k_{BS}.$

Notice that the idealisation step not only requires the understanding of the working of the protocol, but it also demands announcing the corresponding beliefs of a principal at the time of sending its messages. As in the above, the first message is idealised such that $A \xleftrightarrow{k_{AB}} B$ (A's belief about k_{AB}) replaces B, k_{AB}. Similarly, second message attaches the S's belief about k_{AB}. Also note that plaintext messages (A in the first message) are not part of the idealisation.

Step 2: The initial assumptions are stated as follows. Burrows et al. (1990) calls some of the assumptions as dubious as explained in Step 4.

$A \ believes \ A \xleftrightarrow{k_{AS}} S \qquad B \ believes \ B \xleftrightarrow{k_{BS}} S$

$S \ believes \ A \xleftrightarrow{k_{AS}} S \qquad S \ believes \ B \xleftrightarrow{k_{BS}} S$

$S \ believes \ fresh(T_a) \qquad B \ believes \ fresh(T_s)$

$B \ believes \ A \ controls \ A \xleftrightarrow{k} B$

$B \ believes\,(S \ contols\,(A \ believes \ A \xleftrightarrow{k} B)).$

Step 3: Assertions can be easily written using Step 1 as follows.

$$S \ received \ \{T_a, (A \xleftrightarrow{k_{AB}} B)\}k_{AS}$$

$$B \ received \ \{T_s, A \ believes (A \xleftrightarrow{k_{AB}} B)\}k_{BS}.$$

Step 4: Now the protocol analysis is almost trivial as we simply need to apply the BAN rules using the assumptions and the assertions stated above. We omit the detailed derivation steps and write the conclusion as follows.

$$S \ believes \ A \ believes \ A \xleftrightarrow{k_{AB}} B$$

$$A \ believes \ A \xleftrightarrow{k_{AB}} B$$

$$B \ believes \ A \xleftrightarrow{k_{AB}} B$$

$$B \ believes \ A \ believes \ A \xleftrightarrow{k_{AB}} B.$$

Observe that the above derivation became possible when we used some strange assumptions in Step 2. The most dubious assumption is that *B* believes *A* to generate good keys in $B \ believes \ A \xleftrightarrow{k_{AB}} B$ as also mentioned in Burrows et al. (1990). The authors in Burrows et al. (1990) claim a protocol to be secure in BAN logic only if it abides by the assumptions taken during the analysis. That is why, the use of this protocol is restricted to only those scenarios where *A* represents a trusted and competent authority on generating good session keys.

So far, we have presented a brief account on the logic of belief and its application. Now we present some of the extensions of BAN logic.

2.2 BAN extensions

With the emergence of BAN, researchers have suddenly realised the potential of applying logic-based techniques for the formal verification of authentication protocols. Even though BAN has successfully identified flaws in some well-known protocols (Burrows et al., 1990), it has been rigorously analysed for potential weaknesses and several corrective measures have been suggested. The immediate successor of BAN was a logic by Gong et al. (1990) by the name GNY logic. One of the improvements in GNY was to introduce the notion of *recognisability* which captures the recipient's expectation of the contents of a message before actually receiving it. For example, a principal may recognise a particular structure of a message or any form of redundancy in the message. This is in contrast to BAN which assumed that redundancy is always present in encrypted messages. GNY also introduced the notion of *not-originated-here* to identify if a principal receives his own conveyed messages. In addition to extending the applicability of BAN to a wider range of protocols, GNY also separated the notion of possession and beliefs. This allowed one to treat content of a message and the information implied by a message separately because a principal may possess a value but may not believe in it.

Abadi and Tuttle (1991) contributed towards providing new semantics to the logic of BAN. In search of providing a sound semantics, they claimed to have identified many sources of confusion in the original work of BAN. For instance, the authors have given possible-world definition of 'belief' as a form of resource-bounded, de-feasible knowledge. They have reformulated the BAN logic, called AT logic, and proved their axiomatisation to be sound with respect to their model of computation and semantics. The AT logic was closer to traditional modal logics than BAN (Syverson and Cervesato, 2001).

van Oorschot (1993) extended the GNY logic to reason about protocols that involve Diffie-Hellman type key agreement. It was called VO logic. It can be seen that various successors of BAN logic tried to extend BAN in various aspects. Observing this diversity in the BAN suite of logics, the need was felt to come up with a logic which should be sound with respect to its model and which could unify its predecessors. SVO logic of Syverson and Oorschot (1994) is such a logic which unifies four of its predecessors, BAN, GNY, AT, and VO. Next, we briefly describe the SVO logic.

2.3 The logic of SVO

The aim of the SVO logic was to unify four of its predecessors in a sound (with respect to its computational model) way. We briefly describe the SVO logic given in Syverson and Cervesato (2001) as follows.

SVO notations

In addition to the BAN notations as describe above, SVO uses the following notations.

$\neg\varphi$:	SVO added the negation of formulae into its language.
A says X:	It represents whatever is said in the current run of a protocol.
A has X:	It represents all the initial messages of *A* plus all the messages that *A* has received, freshly generated, or can construct using these messages.
$PK(A, k)$:	SVO represents BAN notation for public keys $(\overset{k}{\mapsto} A)$ by $PK(A, k)$.

Furthermore, it splits it into three different kinds of public-keys as $PK_\psi(A, k)$, $PK_\sigma(A, k)$, and $PK_\delta(A, k)$. $PK_\psi(A, k)$ represents public ciphering key of *A*. Only *A* can read messages encrypted with *k*. $PK_\sigma(A, k)$ represents public signature key of *A*. *k* verifies that a message signed by k^{-1} is from *A*. $PK_\delta(A, k)$ represents public key-agreement key of *A*. A Diffie-Hellman key formed with *k* is shared with *A*. For a detailed account on Diffie-Hellman key agreement, see Diffie and Hellman (1976).

$\lfloor X \rfloor_k$, $\{X\}_k$:	SVO separately represents signatures $\lfloor X \rfloor_k$ and encryptions $\{X\}_k$. When used in signatures, *k* basically represents a private-key in $\lfloor X \rfloor_k$.
$\langle X \rangle_{*A}$:	It is used if *A* can not recognise a message (e.g., $\{X\}_k$ if *A* does not know *k*). However, *A* will recognise $\langle X \rangle_{*A}$ as the same thing if received again even if it can not decrypt the message.
X from A:	To represent a message is coming from *A*.

SVO inference rules

Like AT logic, SVO uses only two rules, Modus Ponens and Necessitation, as its inference rules given as follows.

Modus Ponens: From $\vdash\varphi$ and $\varphi \rightarrow \psi$ infer $\vdash\psi$.

Necessitation: From $\vdash\varphi$ infer $\vdash A$ *believes* φ.

Axioms of the logic are all instances of tautologies of classical propositional calculus, and all instances of the following axiom schemata.

SVO axioms

Belief axioms. The following are the classic axioms of modal logic.

1 $(A\ believes\ \varphi \wedge A\ believes\ (\varphi \rightarrow \psi)) \rightarrow A\ believes\ \psi$

2 $A\ believes\ \varphi \rightarrow \varphi$

3 $A\ believes\ \varphi \rightarrow A\ believes\ (A\ believes\ \psi)$

4 $\neg(A\ believes\ \varphi) \rightarrow A\ believes\ (\neg\ A\ believes\ \psi)$.

These axioms represent that a principal believes the logical consequence of its beliefs, a principal's beliefs are always true, a principal can tell what it believes, and a principal can also tell what it does not believe.[4]

Source association axioms. These axioms associate a principal who is responsible for sending an encrypted/signed message. It is called message meaning rule in BAN.

5 $(A \xleftrightarrow{k} B \wedge C\ received\ \{X\ from\ B\}_k) \rightarrow (B\ said\ X \wedge B\ has\ X)$

6 $(PK_\sigma(A, k) \wedge B\ received\ X \wedge SV(X, k, Y)) \rightarrow A\ said\ Y.$

Note that here 'believes' operator is separated from the axiom. Moreover, in the axiom for public-keys, $SV(X, k, Y)$ means that applying k to X confirms that X is the result of signing Y with a private cognate of k.

Key agreement axioms. This axiom captures Diffie-Hellman like key agreement. Diffie-Hellman key agreement is an important component in widely used authenticated key established protocols such as IETF standard Internet Key Exchange (IKE) protocol (Doraswamy and Harkins, 1999). The axiom states that session keys that are the result of good key-agreement keys are good.

7 $(PK_\delta(A, k_A) \wedge PK_\delta(B, k_B)) \rightarrow A \xleftrightarrow{F_0(k_A,\, k_B)} B$

8 $\varphi \equiv \varphi[F_0(k, k')/F_0(k', k)]$.

$F_0(k', k)$ represents function that combines k' with k^{-1} to form a shared key.

Receiving axioms. These axioms state that the concatenates of a concatenated message and the contents of an encrypted or a signed message are also viewed as received messages.

9 $A\ received\ (X_1, \ldots, X_n) \rightarrow A\ received\ X_i$, for $i = 1, \ldots, n$.

10 $(A\ received\ \{X\}_k \wedge A\ has\ k^{-1}) \rightarrow A\ received\ X$

11 $(A\ received\ \lfloor X \rfloor_k) \rightarrow A\ received\ X.$

Axiom 11 assumes that principals possess public keys.

Possession axioms. A principal possesses all of its received messages plus any message obtained as a result of applying any computable function (e.g., encryption, signatures, etc.) on existing messages.

12 $A \textit{ received } X \rightarrow A \textit{ has } X$

13 $A \textit{ has } (X_1, \ldots, X_n) \rightarrow A \textit{ has } X_i$, for $i = 1, \ldots, n$

14 $(A \textit{ has } X_1 \wedge \ldots \wedge A \textit{ has } X_n) \rightarrow A \textit{ has } F(X_1, \ldots, X_n)$.

Comprehension axiom. This axiom basically represents recognisability axiom of GNY (Gong et al., 1990). That is, a principal recognises a function of a message only if he knows the message itself.

15 $A \textit{ believes } (A \textit{ has } F(X)) \rightarrow A \textit{ believes } (A \textit{ has } X)$.

Saying axioms. The following axioms holds a principal responsible for saying each component of a concatenated message. A principal who recently says *X* has said *X*.

16 $A \textit{ said } (X_1, \ldots, X_n) \rightarrow A \textit{ said } X_i \wedge A \textit{ has } X_i$, for $i = 1, \ldots, n$.

17 $A \textit{ says } (X_1, \ldots, X_n) \rightarrow (A \textit{ said } (X_1, \ldots, X_n) \wedge A \textit{ says } X_i)$, for $i = 1, \ldots, n$.

Freshness axioms. These axioms state that a message containing any fresh component is also fresh.

18 $\textit{fresh } (X_i) \rightarrow \textit{fresh } (X_1,\ldots, X_n)$, for $i = 1, \ldots, n$.

19 $\textit{fresh } (X_1, \ldots, X_n) \rightarrow \textit{fresh } F(X_1,\ldots, X_n)$, where F is any computable function which depends upon all of its arguments.

Jurisdiction axiom. A principal having authority on some messages is always right about those messages.

20 $(A \textit{ controls } \varphi \wedge A \textit{ says } \varphi) \rightarrow \varphi$.

Nonce verification axiom. Something if said in the past such that it is fresh, is as if it is said in the current run of a protocol.

21 $(\textit{fresh } (X) \wedge A \textit{ said } X) \rightarrow A \textit{ says } X$.

Symmetric goodness axiom. Symmetric-keys are equivalently good between two principals as follows.

22 $A \xleftrightarrow{k} B \equiv B \xleftrightarrow{k} A$.

Protocol analysis

SVO also analyses a protocol using somewhat similar steps as in BAN except that it does not idealise a protocol in its first step. Rather, it uses the following steps for its analysis.

1 Write initial assumptions.

2 Write all the assertions (as done in Step 3 in BAN).

3 Assert comprehension of received messages. Basically it amounts to adding 'believes' to each received message. Additionally a principal adds $\langle X \rangle_{*A}$ to represent the messages it can not recognise at first.

4 Assert interpretation of the comprehended message from previous step. This basically replaces Step 1 of BAN analysis. Delaying the interpretation until Step 4 (instead of Step 1 in BAN) basically puts the focus on the receiver of a message that how he interprets it, instead of interpreting the understanding of the sender of the message. This reduces the potential for problems that was highly criticised in BAN (Syverson and Cervesato, 2001).

5 This is the last step in which the logic is used to derive beliefs of principals.

We will use this background information on SVO logic to derive axioms in the Distributed Temporal Protocol Logic. In the next section, we briefly introduce the DTPL.

3 The Distributed Temporal Protocol Logic (DTPL)

We briefly introduce the DTPL of Caleiro et al. (2005a), a version of the distributed temporal logic DTL, to reason about protocols and their properties. In DTPL, a distributed system is viewed as a collection of communicating sequential objects (principals or agents) that interact by exchanging messages through an insecure channel *Ch.* A security protocol comprises of a sequence of messages sent and received by two or more principals in a distributed system. DTPL represents a protocol by capturing the temporal aspect of the sequence of actions of each principal executing the protocol. We usually represent by **A** the set of messages that principals communicate where we refer to the elements of **A** as *terms* or *messages.* Moreover, we assume **A** is *freely* generated from two disjoint sets, T (for texts, e.g., nonces or principal ids) and K (for keys). Most of the work in this area uses the assumption of *algebra freeness.* It simply says that two syntactically different terms can not represent the same message. In particular if M, M', N, N' represent terms and k and k' are keys, then according to the algebra freeness assumption:

1 $\{M\}_k = \{M'\}_{k'} \Rightarrow M = M' \wedge k = k'$.

2 $MN = M'N' \Rightarrow M = M' \wedge N = N'$.

3 $MN \neq \{M'\}_k$.

Local alphabet of each principal A comprises of actions Act_A and state propositions $Prop_A$. Act_A includes operations such as *send*(M, B), *rec*(M), *spy*(M), *nonce*(N), and *key*(k) whereas $Prop_A$ includes only *knows*(M). Channel's actions Act_{Ch} include *in*(M, A), *out*(M, A), and *leak* and there is no state proposition associated with the channel.

The *global language* and the *local language* of the logic are defined by the grammar:[5]

$$L ::= @_i[L_i] \mid \bot \mid L \Rightarrow L$$

$$L_i ::= Act_i \mid Prop_i \mid \bot \mid L_i \Rightarrow L_i \mid L_i \,\mathsf{U}\, L_i \mid L_i \,\mathsf{S}\, L_i \mid j : L_j$$

where

- $i, j \in Princ$, a set of principal's ids
- U and S are temporal operators *until* and *since*
- $@_i\,[\phi]$ means that ϕ holds at the current local state of principal i
- $j : \phi$ appearing inside a formula in L_i is called a *communication formula*. It means that principal i has just communicated with principal j for whom ϕ held.

Due to the concurrent nature of the distributed system, event structures are used instead of Kripke structures as the interpretation structures in DTPL. In particular, the interpretation structure $\mu = \langle \lambda, \alpha, \pi \rangle$ of L are suitably labelled distributed life-cycles, built upon a simplified form of Winskel's *event structures* (Winskel, 1987). If Ev_i represents a discrete, linearly ordered, set of events for each principal $i \in Id$:

- λ: is a distributed life-cycle. That is, a prime event structure without conflict, built form Ev_i.
- α_i: $Ev_i \to Act_i$ associates a local action to each local event.
- π_i: $\Xi_i \to \wp(Prop_i)$ associates a set of local state propositions to each *local configuration* ξ_i in the set of local configurations Ξ_i.

Here, *local configuration* of a principal i means a collection of all the local events that have occurred up to a given point. In other words, local configuration of a principal i is a finite set $\xi_i \subseteq Ev_i$ closed under local causality. That is, if $\to_i$ represents the local successor relation between the events in Ev_i, $e \to_i^* e'$, and $e' \in \xi_i$ then also $e \in \xi_i$. Every non-empty local configuration ξ_i is reached by the occurrence of an event $last(\xi_i)$ from the local configuration $\xi_i \backslash last(\xi_i)$. A *global configuration* is a finite set $\xi \subseteq Ev$ closed under global causality, that is, if $e \to^* e'$, and $e' \to \xi$ then also $e \to \xi$. Every global configuration ξ includes the local configuration $\xi|_i = \xi \cap Ev_i$ of each principal i. Moreover, given $e \in Ev$, $e \downarrow = \{e' \in Ev | e' \to^* e\}$ is always a global configuration. The distributed life-cycle of three principals A, B, and C is shown in Figure 2 where dotted vertical line represents communication point. The progress of principal A in shown in Figure 3.

Figure 2 A distributed life-cycle for principals A, B, and C

$$\begin{tikzcd}
A & e_1 \arrow[r] & e_4 \arrow[rr] & & e_7 \arrow[r] & \cdots \\
B & e_2 \arrow[rr] & & e_6 \arrow[d, dash, dotted] \arrow[rr] & & \cdots \\
C & e_3 \arrow[r] & e_5 \arrow[r] & e_6 \arrow[rr] & & \cdots
\end{tikzcd}$$

Figure 3 The progress of principal A

$$\pi_A(\emptyset) \xrightarrow{\alpha_A(e_1)} \pi_A(\{e_1\}) \xrightarrow{\alpha_A(e_4)} \pi_A(\{e_1, e_4\}) \xrightarrow{\alpha_A(e_7)} \pi_A(\{e_1, e_4, e_7\}) \ldots\ldots$$

Using the interpretation structure defined above, the global satisfaction relation at a global configuration ξ of μ can be defined as:

- $\mu, \xi \Vdash @_i[\phi]$ if $\mu, \xi|_i \Vdash_i \phi$
- $\mu, \xi \nVdash \bot$
- $\mu, \xi \Vdash \gamma \Rightarrow \delta$ if $\mu, \xi \nVdash \gamma$ or $\mu, \xi \Vdash \delta$

where the local satisfaction relations $\Vdash_i$ at local configurations are defined as:

- $\mu, \xi_i \Vdash_i act$ if $\xi_i \neq \emptyset$ and $\alpha_i(last(\xi_i)) = act$
- $\mu, \xi_i \Vdash_i p$ if $p \in \pi_i(\xi_i)$
- $\mu, \xi_i \nVdash_i \bot$
- $\mu, \xi_i \Vdash_i \varphi \Rightarrow \psi$ if $\mu, \xi_i \nVdash_i \varphi$ or $\mu, \xi_i \Vdash_i \psi$
- $\mu, \xi_i \Vdash_i \varphi \, \mathsf{U} \, \psi$ if there exists $\xi_i'' \in \Xi_i$ with $\xi_i \subsetneq \xi_i''$ such that $\mu, \xi_i'' \Vdash_i \psi$, and $\mu, \xi_i' \Vdash_i \phi$ for every $\xi_i' \in \Xi_i$ with $\xi_i \subsetneq \xi_i' \subsetneq \xi_i''$
- $\mu, \xi_i \Vdash_i \varphi \, \mathsf{S} \, \psi$ if there exists $\xi_i'' \in \Xi_i$ with $\xi_i'' \subsetneq \xi_i'$ such that $\mu, \xi_i'' \Vdash_i \psi$, and $\mu, \xi_i' \Vdash_i \phi$ for every $\xi_i' \in \Xi_i$ with $\xi_i'' \subsetneq \xi_i' \subsetneq \xi_i$
- $\mu, \xi_i \Vdash_i j : \phi$ if $\xi_i \neq \emptyset$, $last(\xi_i) \in Ev_j$ and $\mu, (last(\xi_i) \downarrow)|_j \Vdash_j \varphi$.

The interpretation structure μ is called a *model* of $\Gamma \subseteq L$ if $\mu, \xi \Vdash \gamma$ for every global configuration ξ of μ and every $\gamma \in \Gamma$. Other operators are defined in Table 1.

Table 1 Temporal operators

Operator	*Meaning*
$\mathsf{X}\,\varphi \equiv \bot \,\mathsf{U}\, \varphi$	Next
$\mathsf{Y}\,\varphi \equiv \bot \,\mathsf{S}\, \varphi$	Previous
$\mathsf{F}\,\varphi \equiv \top \,\mathsf{U}\, \varphi$	Sometime in the future
$\mathsf{P}\,\varphi \equiv \top \,\mathsf{S}\, \varphi$	Sometime in the past
$\mathsf{G}\,\varphi \equiv \neg\, \mathsf{F} \neg\varphi$	Always in the future
$\mathsf{H}\,\varphi \equiv \neg\, \mathsf{P} \neg\varphi$	Always in the past
$\dagger \equiv \neg\, \mathsf{X} \top$	In the end
$* \equiv \neg\, \mathsf{Y} \top$	In the beginning
$\mathsf{F}_o\,\varphi \equiv \varphi \vee \mathsf{F}\,\varphi$	Now or sometime in the future
$\mathsf{P}_o\,\varphi \equiv \varphi \vee \mathsf{P}\,\varphi$	Now or sometime in the past
$\mathsf{G}_o\,\varphi \equiv \varphi \wedge \mathsf{G}\,\varphi$	Now and always in the future
$\mathsf{H}_o\,\varphi \equiv \varphi \wedge \mathsf{H}\,\varphi$	Now and always in the past

In DTPL, the principals are equipped with two functions *synth* and *analz* to compose (through concatenation[6] and encryption) and decompose (through separation and decryption) messages respectively. In particular, if S is a set of messages, then:

The function $synth(S)$ is the smallest set containing S such that:

If $M \in synth(S)$ and $k \in synth(S)$, then $\{M\}_k \in synth(S)$.

If $M_1 \in synth(S)$ and $M_2 \in synth(S)$, then $M_1M_2 \in synth(S)$.

Similarly, $analz(S)$ is the smallest set containing S such that:

If $\{M\}_k \in analz(S)$ and $k^{-1} \in analz(S)$, then $M \in analz(S)$.

If $M_1M_2 \in analz(S)$ then $M_1 \in analz(S)$ and $M_2 \in analz(S)$.

In Caleiro et al. (2005b), a number of axiom schemas have been proposed to represent the specifications of the communication network. The following axiom schemas represent the notion of perfect cryptography. That is, knowledge of each principal only depends on his initial knowledge and on the actions that have occurred.

(K1) $@_A[knows(M_1M_2) \Leftrightarrow (knows(M_1) \wedge knows(M_2))]$,

(K2) $@_A[knows(M) \wedge knows(k) \Rightarrow knows(\{M\}_k)]$,

(K3) $@_A[knows(\{M\}_k) \wedge knows(k^{-1}) \Rightarrow knows(M)]$,

(K4) $@_A[knows(M) \Rightarrow \mathsf{G}_o\, knows(\mathrm{M})]$,

(K5) $@_A[rec(M) \Rightarrow knows(M)]$,

(K6) $@_A[spy(M) \Rightarrow knows(M)]$,

(K7) $@_A[nonce(N) \Rightarrow knows(N)]$, and

(K8) $@_A[key(k) \Rightarrow knows(k)]$.

The first three axiom schemas state that a principal can separate a concatenated message (K1) and encrypt or decrypt a message with the known keys (K2, K3). K4 simply says that a principal does not forget its known messages. K5–K8 say that a principal knows its received, spied, or generated (nonces, keys) messages.

The behaviour of the channel and the way principals communicate with each other are captured in terms of the following axiom schemas.

(C1) $@_{Ch}[in(M, A) \Rightarrow \vee_{B \in Princ}\, B : send(M, A)]$,

(C2) $@_{Ch}[out(M, A) \Rightarrow \mathsf{P}\; in(M, A)]$,

(C3) $@_{Ch}[out(M, A) \Rightarrow A : rec(M)]$,

(P1) $@_A[send(M, B) \Rightarrow \mathsf{Y}(knows(M) \wedge knows(B))]$,

(P2) $@_A[send(M, B) \Rightarrow Ch : in(M, B)]$,

(P3) $@_A[rec(M) \Rightarrow Ch : out(M, A)]$,[7]

(P4) $@_A[spy(M) \Rightarrow Ch : (leak \wedge \mathsf{P} \vee_{B \in Name}\, in(M, B))]$, (P5) $@_A[\wedge_{B\in Princ\setminus\{A\}} \neg B : \top]$,

(P6) $@_A[nonce(N) \Rightarrow \neg Ch : \top]$, and

(P7) $@_A[key(k) \Rightarrow \neg Ch : \top]$.

C1–C3 state that a message arrives at a channel only if it is sent by some principal, that the channel delivers a message only if it was previously arrived, and that a principal receives a message if channel delivers it. The axiom schemas P1–P7 state that a principal must know the message and the recipient of a sending message (P1), that a sending or a receiving message must go through the channel (P2, P3), that a spied messages must have been received and leaked by a channel (P4), that principals do not interact directly (P5), and that nonce and key are not channel events (P6, P7). Finally, the following axiom schemas capture the freshness and uniqueness of the nonces respectively:

(N1) $@_A[nonce(N) \Rightarrow \mathsf{Y}\neg knows(M_N)]$, and

(N2) $@_A[nonce(N)] \Rightarrow \wedge_{B\in Princ\setminus\{A\}} @B[\neg knows(M_N)]$,

where M_N ranges over all the messages containing the nonce N. N1 captures freshness of N as a principal does not know any message containing N before it generates N using *nonce* action. N2 captures uniqueness of N as at the time N is generated, no principal knows any message containing N except the one who generated it.

4 Developing a proof-based verification framework

In this section, we first describe some of the similarities and the differences in the language of messages and formulae in SVO and DTPL. We also describe how the messages and formulae in SVO can be represented in DTPL. Finally, we demonstrate how to benefit from the SVO axioms in order to develop a framework in DTPL that can be used to verify authentication protocols. In doing so, we have tried to refrain from introducing new symbols and constructs into DTPL and used the existing framework as much as possible. However, we do introduce some new notions, and the associated symbols and constructs, either if it was lacking in DTPL or if it considerably simplifies the protocol analysis. Throughout the discussion, we also give comparative comments at appropriate points on how our work differs from the SVO.

4.1 The language of messages and formulae

Both approaches use similar primitives to define their language of messages of security protocols. In particular, SVO assumes the existence of a set T of primitive terms for principal's names, their keys and constants. Whereas, DTPL uses finite sets *Princ* for principal names, *Nonce* for random numbers, and *Key* for keys. Moreover, in the SVO, messages and formulae are built by mutual induction, whereas DTPL treats messages and atomic propositions separately. In both approaches, concatenation and encryption on existing messages introduce new messages.

SVO uses primitive proposition constants, some standard operators, and higher-level constructs (like *believe, sees, has*) to define formulae. In the DTPL, the principals as well as the channel are associated with local actions (Act_A and Act_{Ch}) and local state propositions ($Prop_A$ and $Prop_{Ch}$) that contribute towards formula. DTPL also uses temporal operators given in Table 1 besides standard operators. Below we describe how the logical constructs of SVO can be represented in the DTPL.

- *A received X*. In SVO, it includes all the received messages plus all the messages that can be analysed using *X* by a principal *A*. We do not call the analysed sub-messages of a received message *X* to have been received in DTPL. For example, if $\{M\}_k$ is received by a principal who possesses k^{-1}, then we do not call *M* to be a received message. Instead, we use the notion of *knows* along with *rec* to represent knowing a received message *X* and extend *knows* to all the sub-messages of *X*, by applying the function *analz*, that can be analysed using *X*. It seems more intuitive as *A* did not explicitly receive any of the sub-messages of *X*, but knows them by applying the function *analz*.
- *A sees X*. It includes all the messages that are received, newly generated, or initially available to *A* plus all the messages that *A* can produce from these messages. In effect, the DTPL's actions *rec*, *spy*, *nonce*, *key* and proposition *knows* can be collectively used to define the SVO's *sees* as axiomatised in the axioms K1–K8 excluding K4. Obviously, *sees* does not imply future assertions and hence K4 is not catered in *sees.* If *X* is restricted to only keys *k*, then *A has k* in SVO can be defined in the similar way.
- *A said X*. It simply represents that *A* has sent *X* at some time in the past. Since DTPL has a rich set of operators to capture various temporal aspects, it can easily represent such constructs. In particular, the DTPL's past time operator P_o on its action *send* provides similar effect. Moreover, instead of SVO's two constructs *said* and *says*, we can choose relevant temporal operator on *send* in order to provide a precise interpretation of when a message was exactly said in DTPL.

Now we introduce the following logical constructs into the language of the DTPL.

- *fresh*(*X*). Freshness condition in SVO represents what has not been said prior to the current run of a protocol. Notice that the definition of freshness in SVO provides sufficient condition which not only caters the freshness of nonces, but it can also be applied, in general, on any term. For example, principal's ids are not generally considered fresh because they were transmitted to their owners or to other principals at some time in the past (and the SVO's definition of freshness holds). However, we argue that this definition of freshness can not be generalised for any message. Rather, it should be restricted to only atomic terms (like nonces). For example, according to this definition, provided that an id *A* and a key *k* are not fresh, the message $\{A\}_k$ will still be called fresh just because if nobody ever sent this message in the past. Since freshness is used to capture, generally with the help of a nonce or a timestamp, that a message is communicated recently in a protocol, calling $\{A\}_k$ fresh violates the purpose. Of course, $\{A\}_k$ does not guarantee that a message is fresh. Using the similar idea, DTPL restricts its definition of freshness to the nonces and adopts a more conservative approach by assuming that none of the principals know a fresh message before the time when it was generated.

- $A \xleftrightarrow{k} B$, **PK**(A, k). These key association constructs of SVO provide useful information about a principal's association of certain keys. $A \xleftrightarrow{k} B$ indicates that k is a key exclusively shared between A and B whereas **PK**(A, k) indicates that k is the public-key of A. We extend the DTPL message structure by adopting key association constructs of SVO. However, we change the SVO notation as follows.[8] $\mathbf{PK}_{\psi}(A, k)$ is changed *to* $A \overset{\psi}{\mapsto} k$ (for encryption keys), $\mathbf{PK}_{\sigma}(A, k)$ is changed to $A \overset{\sigma}{\mapsto} k$ (for signature keys), and $\mathbf{PK}_{\delta}(A, k)$ is changed to $A \overset{\delta}{\mapsto} k$ (for key-agreement keys). Moreover, due to the specific nature of these constructs, special meaning can be assigned to them. Specifically, $A \xleftrightarrow{k} B$ *or* $A \overset{x}{\mapsto} k$ also represent a proposition which can be assigned truth values. That is why, they represent both, messages and formulae, the proper use of which can be easily understood from the context.
- *A controls X.* It represents *A*'s jurisdiction over *X.* Since it is normally used to mention authority of a key-server to generate trusted keys, it represents an essential part of authentication protocols. We introduce a proposition *controls*(*X*) to represent this feature in DTPL.

4.2 The axioms of the framework

First, we modify the way DTPL models the communication channel as axiomatised in C1–C3. DTPL models a lossy channel which can lose the data without any intruder intervention. Since our focus is on guaranteeing the security of a protocol with respect to the intruder, we can restrict ourself to a reliable channel. That is, in the absence of an intruder, messages will reach to their intended destination. We add the following axioms to capture the reliability of the communication channel.

(C4) $@_{Ch}[in(M, A) \Rightarrow \mathsf{F}(leak \vee out(M, A))]$

(C5) $@_{Ch}[leak] \Rightarrow @_{A}[spy(M)]$.

By P2, C4, C5, C3, now it is easy to see the following:

(P8) $@_{A}[send(M, B)] \Rightarrow \vee_{P \in Princ}\, @_{P}[\mathsf{F}\,(spy(M) \vee rec(M))]$.

The above axioms state that a channel always transfer the data to a principal, legitimate or otherwise.

DTPL utilises temporal dimension of knowledge of a principal and does not use the notion of 'belief, thereby avoiding confusion caused by different interpretations of the notion of belief in previous logics (Abadi and Tuttle, 1991). In fact, one of the main contributions of GNY (Gong et al., 1990) and its successors (like SVO) was to separate the notion of belief from other notions, like knowing that a shared-key exists between two principals A and $A \xleftrightarrow{k} B$. Below we describe how we take advantage from SVO in order to design axioms for our verification framework of DTPL. We have changed the order in which SVO axioms appear in Section 2 in order to obtain continuity of presentation in this section. We use modus ponens as used by SVO as the only inference rule.

(*MP*) $\varphi \wedge (\varphi \Rightarrow \psi) \Rightarrow \psi$.

Since DTPL does not utilise the epistemic properties of its knowledge (Caleiro et al., 2005a), the necessitation rule and the belief axioms 1–4 of SVO have not been used in our exposition. Moreover, due to their importance, the *source association* axioms of SVO are treated separately in the next subsection.

Possession

Since the notion of *knows* in DTPL represents possession in the SVO, all the possession axioms can be seen in terms of knowledge axioms already described in the previous section. In particular, Axiom 12 of SVO can be represented by Axiom K5 of DTPL and Axiom 13 of SVO can be easily derived using Axiom K1 of DTPL as follows.

(K1a) $@_A[knows(X_1 \ldots X_n) \Rightarrow knows(X_i)]$ for $i = 1, \ldots, n$.

Similarly, Axiom 14 of SVO is the generalised form of K2 and K3 of DTPL given as follows.

(K2a) $@_A[knows(X_1) \wedge \ldots \wedge knows(X_n) \Rightarrow knows(F(X_1 \ldots X_n))]$

where F represents any function computable by A, e.g., encryption, signature etc.

Receiving

All the receiving axioms of the SVO can be translated into DTPL using knowledge Axioms K5, K1 and K3. K5 along with K1a represent Axiom 9 as follows.

$@_A[rec(X_1 \ldots X_n) \Rightarrow knows(X_1 \ldots X_n)]$

$@_A[knows(X_1 \ldots X_n) \Rightarrow knows(X_i)]$ for $i = 1, \ldots, n$.

Note that DTPL's equivalent of SVO's Axiom 9 is given in terms of knowing the concatenates of a received concatenated message instead of receiving it. Similarly receiving an encrypted term such that a principal holds the corresponding decryption-key means that the principal knows the contents of the encrypted message. Since we have adopted a richer representation of key association from SVO, we can break down K3 of DTPL for symmetric-key encryption, public-key encryption, and signatures as follows.

(K3a) $@_A[knows(\{M\}_{k_{AS}}) \wedge knows(A \xleftrightarrow{k_{AS}} S) \Rightarrow knows(M)]$,

(K3b) $@_A[knows(\{M\}_k) \wedge knows(A \overset{\psi}{\mapsto} k) \Rightarrow knows(M)]$,

(K3c) $@_A[knows(\{M\}_{k^{-1}}) \wedge knows(B \overset{\sigma}{\mapsto} k) \Rightarrow knows(M)]$.

Notice that SVO's notion of key association not only captures different notions of encryption, but also provides a clear understanding of knowing the relevant key associated to a principal. For example, knowing $A \overset{\psi}{\mapsto} k$ not only implies the possession of k but also binds it with A. The Axiom K5 along with K3(a, b, c) give the DTPL representation of Axioms 10 and 11 of the SVO. Although SVO represents the sub-messages of a received message as being received, we believe that our adherence to the notion of received messages to only those which are actually received during communication is more intuitive.

Comprehension and saying

The way DTPL models knowledge (K1–K8), it can not be claimed that a principal A knows X if he knows $F(X)$. For example, a principal knows a message if he receives it (K5), but he may not be able to invert $F(X)$ to obtain X. As also mentioned in SVO, its Axiom 15 does not imply that F is invertible by A. Therefore, we do not use the comprehension axiom of SVO in DTPL. Moreover, since we do not hold a principal responsible for saying all the concatenates of a concatenated message, we intentionally remove the saying Axiom 16 in our work. However, the formula P1 partially represents Axiom 16 in which a principal must know what it says. We do not have two different temporal constructs for sending a message like SVO's *says* and *said*. Since we can use past time temporal operators to capture several past time activities, we do away with Axiom 17 in our work.

Freshness

Since a nonce action *nonce*(N) in DTPL represents the notion of freshness, we use this action to introduce a new proposition *fresh*(N) similar to SVO. In particular, we introduce:

(F1) $@_A[\mathit{nonce}(N) \Rightarrow \mathit{fresh}(N)]$.

The rules N1 and N2 clarify the freshness concept in the DTPL. As used in SVO, a term remains fresh in the current run of a protocol. That is,

(F1a) $@_A[\mathit{fresh}(N) \Rightarrow \mathsf{X}\ \mathit{fresh}(N)]$.

Now, we can use Axiom 18 of SVO as follows.

(F2) $@_A[\mathit{fresh}(X_i) \Rightarrow \mathit{fresh}(X_1 \ldots X_n)]$, for $i = 1, \ldots, n$.

It should be noted that $\mathit{fresh}(X_1 \ldots X_n)$ means that the entire message $X_1 \ldots X_n$ is fresh. It may be because any of the X_i or all of the X_i are fresh. So given only $\mathit{fresh}(X_1 \ldots X_n)$, one can not tell with certainty that which sub-message is fresh. Similarly any computable function by A of a fresh term is also fresh as given by SVO's Axiom 19.

(F3) $@_A[\mathit{fresh}(X_1 \ldots X_n) \Rightarrow \mathit{fresh}(F(X_1 \ldots X_n))]$.

For the same reason mentioned above, function F must depend on all of its arguments in order to guarantee fresh output.

Jurisdiction

Like SVO's Axiom 20, the following axiom captures the jurisdiction of a principal for generating keys in DTPL.

(J1) $@_S[\mathit{controls}(\varphi_k) \wedge \mathit{send}(M_k, A)] \Rightarrow \vee_{P \in \mathit{Princ}} @_P[\mathsf{F}\ \mathit{knows}(\varphi_k)]$.

That is, if a principal is known to control a formula for generating keys φ_k and he sends a message containing the key k then the receiver knows φ_k to be true. Of course, the receiver does need to make sure that S sent M_k before it concludes. By P8, someone will receive M_k and apply J1 to know φ_k. In order to represent a principal's jurisdiction over generating keys, we restrict φ_k to be of type $A \xleftrightarrow{k} B$ or $A \overset{x}{\mapsto} k$ (x is the key type in public-key system).

Nonce verification

The nonce verification axiom of SVO is used to transform the past (*said*) into present (*says*) using the notion of freshness (*fresh*). DTPL can clearly pin point a past time event using its explicit notion of *configuration.* That is why, DTPL introduces only a single action *send* to represent sending a message and benefits from its temporal operators to capture the exact timing of the event. Therefore, we do not use the nonce verification axiom of SVO.

Key agreement

These axioms of SVO specifically target the protocols involving a key agreement such as Diffie-Hellman key exchange. We use the key agreement axiom of SVO in the following.

$$\text{(K9) } @_A[knows\,(A \overset{\delta}{\mapsto} k_A) \wedge knows\,(B \overset{\delta}{\mapsto} k_B) \Rightarrow knows\,(A \overset{F_0(k_A,k_B)}{\longleftrightarrow} B)]\,.$$

Here, F_0 is some key-agreement function such that $F_0(k_A, k_B) = F_0(k_B, k_A) = k_{AB}$.

Symmetric goodness

We simply use the symmetric goodness axiom of SVO as follows:

$$\text{(G1) } @_C[knows\,(A \overset{k}{\longleftrightarrow} B) \Leftrightarrow knows\,(B \overset{k}{\longleftrightarrow} A)].$$

4.3 *Originators of the received messages*

This subsection is entirely devoted to the *source association axioms* of SVO because of the key role they play in verifying authentication in a protocol. Basically source association pertains to claiming that a message can only be bound to certain source who is responsible for originating that message. Before we actually translate the source association axioms, we introduce the notion of 'origination'. If a principal sends a term in a message such that he never communicated that term in any message in the past then he originates the term in its sending message. The notion of origination is not new, but has also been mentioned in the work of strand spaces in Fabrega et al. (1999). Message origination can be captured with the help of following axiom.

$$\text{(O1) } @_A[send(M_N, B) \wedge \mathsf{H}(\neg send\,(M'_N, C) \wedge \neg rec\,(M'_N) \wedge \neg spy\,(M'_N)) \Leftrightarrow Orig(M_N)].$$

That is, A originates N in M by sending M such that it never communicated any message M' that contained N.

Authentication protocols can be categorised according to different underlying cryptography used for their implementation. They also utilise different mechanisms to achieve authentication. The source association axioms in SVO and in others (like BAN, AT, GNY etc.) utilise the same categorisation of authentication protocols. In particular, SVO ascribes two axioms for source association, one for a received message encrypted by a shared-key of two principals, and other for a received signed message. We briefly mention how authentication protocols can be broadly classified into three categories. We also present our formulation of the source association axioms according to each category in the following.

Symmetric-key protocols

Cryptographic protocols often make use of the symmetric-key encryption in order to achieve their authentication goals. The symmetric-key cryptography has the advantage that an encrypted message contains guarantee about the originator of a received message. That is, an encrypted message can originate only from the principals having access to the encryption key with which the message was encrypted. This is a direct consequence of the assumption of ideal cryptography.[9] Since in symmetric-key cryptography, a key is assumed to be a principal's safe secret, encrypting a message under symmetric-key ensures the possession of the key, and hence the origination of the message by a principal having that key. We axiomatise the source association notion of SVO (Axiom 5) as follows.

$$\text{(O2)}\ @_A[knows(P \xleftrightarrow{k} Q) \wedge rec(\{X\}_k)] \Rightarrow \bigvee_{B\in\{P,Q\}} @_B[\mathsf{P}\ Orig(\{X\}_k)].$$

Since A knows the key k, generally A is either P or Q in the above axiom. Moreover, if a server generated the key k for P and Q, then A may also represent the server. Since a server never encrypts a message with the shared key of principals, B does not represent the server. Notice that the above formula captures the possibility of a message originated by both principals possessing the key k. This distinguishes our formalisation of the notion of source association from others. SVO assumes that a principal can recognise any message that it has seen before using its notion of recognisability in $\langle X\rangle_{*C}$. Therefore, it restricts its attention to only those messages that are known to be coming form a principal other than him by adding *from* with the received message in its axiom, that is, C *received*$\{X\ from\ B\}_k$ in Axiom 5. We make no such assumptions. Our axiomatisation captures those situations also in which a principal's own originated message is redirected back to him in order to launch an attack. See Guttman and Fábrega (2002) for such an attack on Woo-Lam one-way authentication protocol (Woo and Lam, 1992). However, we do assume that A is able to distinguish X from any garbage value. This is necessary otherwise, after decrypting any garbage value Y ($Y = \{X\}_k$), A would not know if the content of Y, i.e., X, was really encrypted with k or was just a random bitstream. Moreover, note that we do not need to include DTPL equivalent of B *has* X (i.e., *knows*(X)) in the consequence of the above rule as done in Axiom 5 of SVO, because according to P1, B already knows what it sends.

Asymmetric-key protocols

In asymmetric-key cryptography, a signed message originates from a principal who has access to the private-key with which the message was signed. We assume the private-key of a principal to be its safe secret. Since principal's *ids* are unique and their mapping to the private-keys are injective in asymmetric cryptography, signing a message by a principal's private-key assures the originator of the signed message. The asymmetric-key equivalent of the SVO source association Axiom 6 can be given as:

$$\text{(O3)}\ @_A[knows(B \overset{\sigma}{\mapsto} k) \wedge rec(\{X\}_{k^{-1}})] \Rightarrow @_B[\mathsf{P}\ Orig(\{X\}_{k^{-1}})].$$

We have avoided using the extra notation introduced in SVO as in the above axiom it is assumed that applying k to $\{X\}_{k^{-1}}$ confirms that $\{X\}_{k^{-1}}$ is the results of signing X with the private-key k^{-1}. Similarly, we avoided using separate notations for encryption and signing as it can be simply understood from the syntax ($\{X\}_k$ represents an encrypted message whereas $\{X\}_{k^{-1}}$ represents a signed message).

Challenge-response protocols

This is the last category that we use to identify the originator of a received message. None of the methods provide any axiom for this category in the entire BAN suite. This is because in the protocols using public-key cryptography, an encrypted message does not provide any guarantee in the identity of its sender. The reason being the obvious fact that anyone can encrypt a message masquerading someone else by using public-key of the intended recipient. That is why, we use a challenge-response strategy that public-key protocols typically rely on in which they use an encrypted secret in their challenge messages in order to achieve authentication (for example, the Needham-Schroeder public-key protocol (Needham and Schroeder, 1978)). If a nonce or a pre-shared secret in an encrypted message is used as a challenge, the correct response may be to generate a reply containing that nonce or the secret. It ensures that the challenge is not only received by the intended principal but it also opened (decrypted) the challenge message in order to retrieve the secret and composed a reply. The challenger may need to check the structure of reply message to ensure that an intruder has not simply forwarded the challenge message back to the challenger (Muhammad et al., 2006).

Since public-key encryption does not provide any information regarding source association, $B \overset{\psi}{\mapsto} k$ can not be directly used in the source association axiom. We use the above challenge-response idea to identify the originator of a received message in public-key protocols as follows. If N is a fresh secret uniquely originated in an encrypted challenge message $\{M_N\}_k$ such that only a principal B has the corresponding private-key to obtain N from $\{M_N\}_k$, then the reception of a message having N in any form other than $\{M_N\}_k$ ensures that B knows $\{M_N\}_k$, decrypted it and released N in any form other than $\{M_N\}_k$. The intuition is that since only B has the decryption key to discover N from $\{M_N\}_k$ and N is originated uniquely (so that no other principal knows N except the one who originated it), reception of any message in which N occurs in any form other than $\{M_N\}_k$ confirms that $\{M_N\}_k$ has been decrypted and the information has been released by B. The idea of treating security protocols as challenge-response protocols is not new. For example, similar idea has been presented, not in the logic-based setting though, in the authentication tests of strand spaces in Guttman and Fábrega (2002).

$$\text{(O4)}\quad @_A\{(\neg send(M'_N, C)\,\mathsf{S}\,(Orig(\{M_N\}_k) \wedge fresh(N))) \wedge rec(M''_N) \wedge knows(B \overset{\psi}{\mapsto} k)\}$$
$$\Rightarrow @_B[\mathsf{P}\,(Orig(M'''_N) \wedge \mathsf{P}\;knows(\{M_N\}_k))]$$

where, N must exist in M''_N in a form other than $\{M_N\}_k$. Moreover, N also exists in M'''_N in a form other than $\{M_N\}_k$. The above axiom is the generalised form of the challenge-response based on public-key encryption. The condition that N must exist in M''_N in a form other than $\{M_N\}_k$ guards against the attack where an intruder simply forwards $\{M_N\}_k$ back to A without involving B at all. Therefore, if this condition is met, B must have received and decrypted $\{M_N\}_k$ and released N in any form other than $\{M_N\}_k$ in M'''_N.

So far, we have presented the extended DTPL framework that can be used to analyse authentication protocols in a proof-based environment. Next, we present how to apply the proposed framework in order to analyse a protocol.

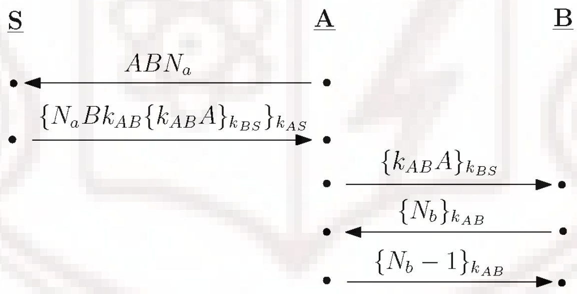
5 Verifying authentication in Needham-Schroeder protocol

The Needham-Schroeder Shared-Key (NSSK) protocol (Burrows et al., 1990) is a well-known protocol that has influenced the design of many authentication protocols (for example, Kerberos protocol developed at MIT was based on it (Miller et al., 1988)). We analyse its authentication property using the proposed axioms.

5.1 *The protocol description*

The NSSK protocol is depicted in Figure 4. The goal of this protocol is to distribute a session key k_{AB} from a trusted server S to two principals A and B. Principals already possess secret shared-keys (k_{AS} and k_{BS}) with the server. N_a and N_b are the nonces of A and B respectively. The protocol begins by A, the initiator, sending its request to S in the first message. Upon receiving the first message, the server generates a session key k_{AB} and sends the second message to A. The initiator A extracts the relevant information and forwards the encrypted sub-message of its message to B. Upon receiving the message, B, the responder, generates a nonce and encrypts it with the received session key and sends it to A. Finally, A replies B back in the last message. A subtracts 1 from B's nonce in order to differentiate the last two messages.

Figure 4 Needham-Schroeder (NSSK) protocol



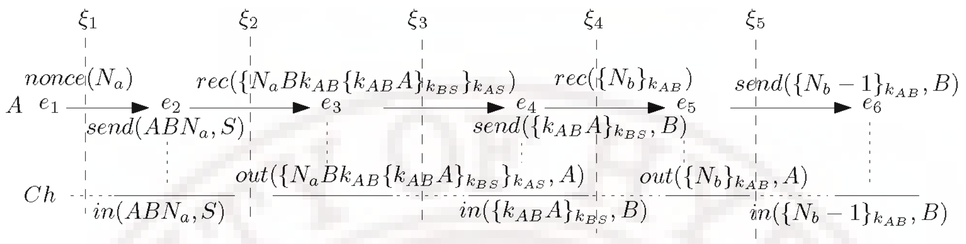
5.2 *The initiator's perspective*

We begin by analysing the protocol from the initiator's perspective. The initiator's sequence of messages can be represented in terms of DTPL as follows.

1. $@_A[send(\{n_B - 1\}_{k_{AB}}, B) \wedge \mathsf{P}\,(rec(\{n_B\}_{k_{AB}}) \wedge \mathsf{P}\,(send(\{k_{AB}A\}_{k_{BS}}, B)$
   $\wedge\, \mathsf{P}\,(rec(\{N_A Bk_{AB}\{k_{AB}A\}_{k_{BS}}\}_{k_{AS}}) \wedge \mathsf{P}\,(send(ABN_A, S) \wedge \mathsf{P}\,nonce(N_a)))))].$

In the above, Y could be used instead of the past time operator P, but we stick to the representation given in Caleiro et al. (2005a). The life-cycle of initiator A is depicted in Figure 5. Notice the vertical dashed lines in the figure indicating different configuration points in the run of the initiator. Each event e_i corresponding to each action of the initiator of the protocol changes A's configuration from ξ_{i-1} to ξ_i.

Figure 5 Life-cycle of principal A in NSSK protocol


ξ_1
ξ_2
ξ_3
ξ_4
ξ_5
$nonce(N_a)$
$rec(\{N_aBk_{AB}\{k_{AB}A\}_{k_{BS}}\}_{k_{AS}})$
$rec(\{N_b\}_{k_{AB}})$
$send(\{N_b-1\}_{k_{AB}}, B)$
A
e_1
e_2
e_3
e_4
e_5
e_6
$send(ABN_a, S)$
$send(\{k_{AB}A\}_{k_{BS}}, B)$
$out(\{N_aBk_{AB}\{k_{AB}A\}_{k_{BS}}\}_{k_{AS}}, A)$
$out(\{N_b\}_{k_{AB}}, A)$
Ch
$in(ABN_a, S)$
$in(\{k_{AB}A\}_{k_{BS}}, B)$
$in(\{N_b-1\}_{k_{AB}}, B)$

The initial set of assumptions related to principal A is as follows:

$@_A[* \Rightarrow knows(A \xleftrightarrow{k_{AS}} S)]\,)$

$@_S[* \Rightarrow knows(A \xleftrightarrow{k_{AS}} S)]$

where $*$ (see Table 1) captures initial configuration of a principal for this protocol. Knowledge is treated in DTPL as non-decreasing as formulated by K4. This monotonicity has been adopted by several researchers, such as BAN and its successors, where they treated belief as non-decreasing. We apply MP and K4 and use the above assumptions to get the following results at any configuration.

A1. $@_A[knows(A \xleftrightarrow{k_{AS}} S)]$

A2. $@_S[knows(A \xleftrightarrow{k_{AS}} S)]$.

Similarly, we also assume the server's authority for generating session keys.

A3. $@_S[controls(A \xleftrightarrow{k} B)]$.

Applying O2, A1, A2, 1, and MP at configuration ξ_3:

2. $@_Q[\mathsf{P}\ Orig(\{N_aBk_{AB}\{k_{AB}A\}_{k_{BS}}\}_{k_{AS}})]$

where $Q \in \{A, S\}$ originates the above message at $\xi \subset \xi_3$ (capturing 'sometime in the past'). Since a principal originating a message sends that message to some principal C, using O1, MP, and 2 at ξ:

3. $@_Q[send(\{N_aBk_{AB}\{k_{AB}A\}_{k_{BS}}\}_{k_{AS}}, C)]$.

For this protocol to work properly, notice that Q should be actually the server S and not the principal A. But in our case, $Q \in \{A, S\}$. So in order to check the possibility if $Q = A$, we use a fairly straight forward method. We generalise the message by focusing on abstract message structure in the protocol without instantiating any message variable to any particular value. That is, Q sends $\{NYk\{kX\}_{k_1}\}_{k_2}$ (here, N is a nonce, X and Y are principal ids, and k, k_1, and k_2 are keys). We simply analyse all the sending events of principal A. We do not check the server because that what we are trying to prove.

Examining all the actions corresponding to sending events and using the assumption of message *algebra freeness*, it is easy to realise that none of the sending action has a message corresponding to $\{NYk\{kX\}_{k_1}\}_{k_2}$. Therefore, we remove the possibility that a message of this form is sent by any principal except *S*. Therefore, *Q* is in fact equal to *S*. From now on, we replace *Q by S*.

Since *S* must know its sending message before sending it. According to P1, MP, and 3 at a configuration $\xi' = \xi \backslash last(\xi)$ (capturing 'previous' Y in P1).

4. $@_S[knows(\{N_a B k_{AB}\{k_{AB}A\}_{k_{BS}}\}_{k_{AS}})]$.

Therefore, *S* having the decryption key also knows the contents of the encrypted term. Using K3a, A2, MP, and 4 at ξ':

5. $@_S[knows(\{N_a B k_{AB}\{k_{AB}A\}_{k_{BS}}\}]$.

Since *A* received $\{N_a B k_{AB}\{k_{AB}A\}_{k_{BS}}\}_{k_{AS}}$ at ξ_3 (from 1) and knows the corresponding key (from A1), it knows the encrypted message as well as its contents. From K5, 1, and MP at ξ_3:

6. $@_A[knows(\{N_a B k_{AB}\{k_{AB}A\}_{k_{BS}}\}_{k_{AS}})]$.

From K3a, A1, MP and 6 at ξ_3:

7. $@_A[knows(N_a B k_{AB}\{k_{AB}A\}_{k_{BS}})]$.

Using K1a, 7 and MP:

8. $@_A[knows(\{k_{AB}A\}_{k_{BS}})]$.

Note that *A*'s knowing $\{k_{AB}A\}_{k_{BS}}$ does not mean knowing its content because *A* does not possess the key k_{BS}. Therefore, $\{k_{AB}A\}_{k_{BS}}$ appears as arbitrary term to *A*. From the analysis so far, it can be deduced that the server is familiar with the initiator's nonce and the responder's id and generated k_{AB}. Now using F1, 1, and MP at ξ_1:

9. $@_A[nonce(N_a) \Rightarrow fresh(N_a)]$.

From N1 and N2 at ξ_1, we can say that no principal ever knew any message before ξ_1 containing N_a as its subterm. Using F2, 9, and MP:

10. $@_A[fresh(N_a B k_{AB}\{k_{AB}A\}_{k_{BS}})]$

and from F3, 10, and MP:

11. $@_A[fresh(\{N_a B k_{AB}\{k_{AB}A\}_{k_{BS}}\}_{k_{AS}})]$.

Therefore, from *A*'s perspective, the server not only generated k_{AB}, but it generated it freshly. Furthermore, *A* also deems k_{AB} as the right session key to be used between him and *B* as stated by J1, 3, A3, and MP:

12. $@_A[knows(A \xleftrightarrow{k_{AB}} B)]$.

So far we have established *A*'s understanding from the message exchanges between him and the server. Now we examine the interaction between principals *A* and *B* from the initiator's perspective. *A* sends $\{k_{AB}A\}_{k_{BS}}$ to *B* at ξ_4 and receives $\{N_b\}_{k_{AB}}$ at ξ_5. Using R2, 1 and MP at ξ_5:

13. $@_A[knows(\{N_b\}_{k_{AB}})]$.

Notice that here we can not apply the origination formula for symmetric keys because even if *A* decrypts $\{N_b\}_{k_{AB}}$, it can not recognise N_b, violating the assumption in O2. From K3a, 13, 12, and MP at ξ_5:

14. $@_A[knows(N_b)]$.

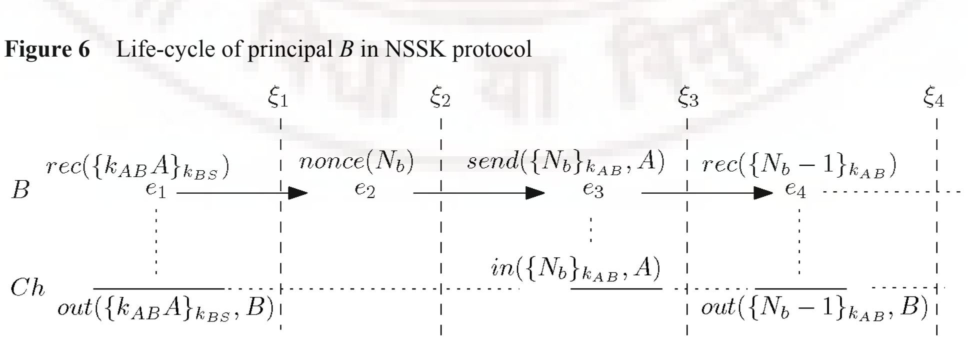
Since *A* had no knowledge of N_b before this point in time and there is no extra information in the message $\{N_b\}_{k_{AB}}$ that may help *A* recognise the correctness of this message, *A* can not conclude anything further. That is, even though *A* possesses k_{AB} and is able to extract N_b from its received message $\{N_b\}_{k_{AB}}$, it can not tell if the nonce N_b is actually the responder's nonce or any garbage value. This suggests a possible weakness in the protocol where an intruder could simply replace $\{N_b\}_{k_{AB}}$ by any random value *X*. As long as *A* does not recognise *X* (using any other means) as an improper message, the intruder can successfully make *A* feel that it has completed the protocol with *B* whereas *B* may not have been involved in the protocol at all. This is because we have not assumed that a principal can tell where a message is coming from by just looking at the structure of the message. That is why, *A* could not recognise N_b from the message $\{N_b\}_{k_{AB}}$. It should be obvious by now that this problem can be resolved by simply adding in the second last message something that *A* can recognise, such as *B*'s id. This problem and the corresponding solution was first suggested in Gong et al. (1990). This concludes our analysis of the protocol from the initiator's side. Next, we analyse the protocol from the responder's perspective.

5.3 *The responder's perspective*

The responder's life cycle is depicted in Figure 6. The responder's sequence of actions can be written as:

1. $@_B[rec(\{N_B - 1\}_{k_{AB}}) \wedge \mathsf{P}\,(send(\{N_B\}_{k_{AB}}, A) \wedge \mathsf{P}\,(nonce(N_b) \wedge \mathsf{P}\,rec(\{k_{AB}A\}_{k_{BS}})))]$.

Figure 6 Life-cycle of principal *B* in NSSK protocol


ξ_1
ξ_2
ξ_3
ξ_4
$rec(\{k_{AB}A\}_{k_{BS}})$
$nonce(N_b)$
$send(\{N_b\}_{k_{AB}}, A)$
$rec(\{N_b - 1\}_{k_{AB}})$
B
e_1
e_2
e_3
e_4
Ch
$in(\{N_b\}_{k_{AB}}, A)$
$out(\{k_{AB}A\}_{k_{BS}}, B)$
$out(\{N_b - 1\}_{k_{AB}}, B)$

The initial set of assumptions is as follows:

$@_B[* \Rightarrow knows(B \xleftrightarrow{k_{BS}} S)]$ and $@_S[* \Rightarrow knows(B \xleftrightarrow{k_{BS}} S)]$.

By MP, K4, and the above assumptions:

A1. $@_B[knows(B \xleftrightarrow{k_{BS}} S)]$

A2. $@_S[knows(B \xleftrightarrow{k_{BS}} S)]$

A3. $@_S[controls(A \xleftrightarrow{k} B)]$.

From O2, A1, A2, 1, and MP at ξ_1:

2. $@_Q[\mathsf{P}\ Orig(\{k_{AB}A\}_{k_{BS}})]$.

Here, $Q \in \{B, S\}$. Using the similar reason as given before and assuming that the message algebra is freely generated, it can be seen that none of the B's sending actions correspond to a message of the form $\{kX\}_{k'}$. Therefore, from O1, MP, and 2 at $\xi \in \xi_1$:

3. $@_S[send(\{k_{AB}A\}_{k_{BS}}, C)]$.

From P1, MP, and 3 at $\xi' = \xi \backslash last(\xi)$

4. $@_S[knows(\{k_{AB}A\}_{k_{BS}})]$.

From K3a, A2, MP, and 4 at ξ':

5. $@_S[knows(k_{AB}A)]$.

From K5, 1, and MP at ξ_1:

6. $@_B[knows(\{k_{AB}A\}_{k_{BS}})]$.

From K3a, A1, MP and 6 at ξ_1:

7. $@_B[knows(k_{AB}A)]$.

Therefore, the responder is aware of the session key originated by the server. Note that unlike the initiator, B can not gain any assurance in the freshness of its received message. Therefore, from B's perspective, although the server has generated k_{AB}, it may not have generated it freshly. Therefore, examining the protocol from the responder's perspective also reveals a vulnerability to a known replay attack. If an attacker records one run of this protocol and subsequently learns the key k_{AB}, he can replay the message $\{k_{AB}A\}_{k_{BS}}$ to B. Being unable to tell that the key k_{AB} is not fresh, B will accept it as a legitimate request to initiate a session using that key. The authors of BAN in Burrows et al. (1990) mentioned this vulnerability in the protocol. That is why, BAN had to resort to the dubious assumption that *B believes fresh* $A \xleftrightarrow{k_{AB}} B$ in order to attain authentication.

B also considers k_{AB} as the right session key to be used between him and A as stated by J1, 3, A3, and MP:

8. $@_B[knows(A \xleftrightarrow{k_{AB}} B)]$.

By F1, 1, and MP at ξ_2:

9. $@_B[nonce(N_b) \Rightarrow fresh(N_b)]$.

By F3, 9, and MP:

10. $@_B[fresh(\{N_b\}_{k_{AB}})]$.

11. $@_B[fresh(\{N_b - 1\}_{k_{AB}})]$.

B receives $\{N_b - 1\}_{k_{AB}}$ at ξ_4. Since *A* can recognise $N_b - 1$, by O2, 8, 1, and MP at $\xi \subset \xi_4$:

12. $@_A[Orig(\{N_b - 1\}_{k_{AB}})]$. .

Since $N_b \neq N_b - 1$ (free message algebra), *B* could not have originated the above message. Therefore, assuming that the session key k_{AB} is fresh, the responder of the protocol guarantees that the initiator shares the same session key with him.

The purpose of the protocol was to distribute a session key between both principals. But analysing the protocol from the initiator's perspective reveals that the initiator is not sure if the responder also possesses the session key at the end of the protocol. Furthermore, the responder is not sure that it shares the fresh session-key with the initiator until it assumes that k_{AB} is always freshly generated.

6 Conclusion

We have presented a proof-based framework for the verification of security protocols using DTPL. For this purpose, we have utilised the framework of SVO, which unifies four of its famous predecessors in a sound way. We have demonstrated how the similarities between both methods helped us utilise the existing SVO axioms in DTPL. We have also clarified some of the notions and extended the framework of SVO in our work. We have categorised authentication protocols as symmetric-key, asymmetric-key and challenge-response protocols. We have formalised the source association axioms of SVO in terms of these categories. Our work contains the advantages of both, the conciseness of the proof-based techniques as well as the clear representation of the temporal aspects of a protocol run of DTPL.

Acknowledgment

This work was partially supported by ARO under grant DAAD 9-01-1-0502. The views and conclusions herein are those of the authors and do not represent the official policies of the funding agencies or the University of Central Florida.

References

Abadi, M. and Tuttle, M. (1991) 'A semantics for a logic of authentication', *Proceedings of the 10th Annual ACM Symposium on Principles of Distributed Computing*, pp.201–216.

Basin, D., Modersheim, S. and Vigano, L. (2005) 'OFCM: a symbolic model checker for security protocols', *International Journal of Information Security*, Vol. 4, No. 3, pp.181–208.

Bodei, C., Degano, P., Focardi, R. and Priami, C. (2002) 'Primitives for authentication in process algebras', *Theoretical Computer Science*, Vol. 283, No. 2, June, pp.271–304.

Burrows, M., Abadi, M. and Needham, R. (1990) 'A logic of authentication', *ACM Transactions on Computer Systems*, Vol. 8, No. 1, February, pp.18–36.

Caleiro, C., Vigano, L. and Basin, D. (2005a) 'Metareasoning about security protocols using distributed temporal logic', *Electronic Notes in Theoretical Computer Science*, Vol. 125, No. 1, pp.67–89.

Caleiro, C., Vigano, L. and Basin, D. (2005b) 'Relating strand spaces and distributed temporal logic for security protocol analysis', *Logic Journal of IGPL*, Vol. 13, No. 6, pp.637–663.

Chellas, B.F. (1980) *Modal Logic: An Introduction*, Cambridge University Press, Cambridge, UK.

Clarke, E.M., Jha, S. and Marrero, W. (2000) 'Verifying security protocols with Brutus', *ACM Transactions on Software Engineering and Methodology* (*TOSEM*), Vol. 9, No. 4, pp.443–487.

Coffey, T., Dojen, R. and Flanagan, T. (2003) 'Formal verification: an imperative step in the design of security protocols', *Computer Networks*, Vol. 43, pp.601–618.

Diffie, W. and Hellman, M. (1976) 'New directions in cryptography', *IEEE Transactions on Information Theory*, Vol. 22, No. 6, pp.644–654.

Dolev, D. and Yao, A.C. (1983) 'On the security of public key protocols', *IEEE Transactions on Information Theory*, Vol. 29, March, pp.198–208.

Doraswamy, N. and Harkins, D. (1999) *Ipsec: The New Security Standard for the Internet, Intranets, and Virtual Private Networks*, Prentice-Hall, Englewood Cliffs, New Jersey, USA.

Ehrich, H-D. and Caleiro, C. (2000) 'Specifying communication in distributed information systems', *Acta Informatica*, Vol. 36, No. 8, pp.591–616.

Fabrega, F.T., Herzog, J. and Guttman, J. (1999) 'Strand spaces: proving security protocols correct', *Journal of Computer Security*, Vol. 7, No. 1, pp.191–230.

Fidge, C. (2001) *A Survey of Verification Techniques for Security Protocols*, Software Verification Research Centre, School of Information Technology, The University of Queensland, Tech. Rep. 01-22, Brisbane, Australia.

Gong, L., Needham, R. and Yahalom, R. (1990) 'Reasoning about belief in cryptographic protocols', *Proceedings of the IEEE Symposium on Research in Security and Privacy*, May, pp.234–248.

Guttman, J.D. and Fábrega, F.J.T. (2002) 'Authentication tests and the structure of bundles', *Theoretical Computer Science*, Vol. 283, June, pp.333–380.

Kemmerer, R. (1989) 'Using formal methods to analyze encryption protocols', *IEEE Journal on Selected Areas in Communications*, Vol. 7, No. 4, pp.448–457.

Longley, D. and Rigby, S. (1992) 'An automatic search for security flaws in key management schemes', *Computers and Security*, Vol. 11, No. 1, pp.75–90.

Lowe, G. (1996) 'Breaking and fixing the Needham-Schroeder public-key protocol using FDR', in Margaria, T. and Steffen, B. (Eds.): *Tools and Algorithms for the Construction and Analysis of Systems, 2nd International Workshop TACAS'96*, Ser. LNCS 1055, Springer-Verlag, Berlin, Germany, March 27–29, pp.147–166.

Marrero, W., Clarke, E. and Jha, S. (1997) 'A model checker for authentication protocols', *DIMACS Workshop on Design and Formal Verification of Security Protocols*, September 3–5, Available online: http://dimacs.rutgers.edu/Workshops/Security/program2/program.html.

Meadows, C. (2003) 'Formal methods for cryptographic protocol analysis: emerging issues and trends', *IEEE Journal on Selected Areas in Communications*, Vol. 21, No. 1, January, pp.44–54.

Meadows, C.A. (1992) 'Applying formal methods to the analysis of a key management protocol', *Journal of Computer Security*, Vol. 1, No. 1, pp.5–36.

Millen, J.K. and Shmatikov, V. (2001) 'Constraint solving for bounded-process cryptographic protocol analysis', *Proceedings of the 8th ACM Conference on Computer and Communications Security*, pp.166–175.

Millen, J.K., Clark, S.C. and Freedman, S.B. (1987) 'The interrogator: protocol security analysis', *IEEE Transactions on Software Engineering*, Vol. 13, No. 2, February, pp.274–288.

Miller, S.P., Neuman, B.C., Schiller, J.I. and Saltzer, J.H. (1988) *Kerberos Authentication and Authorization System*, Project Athena Technical Plan, Section E.2.1.

Muhammad, S., Furqan, Z. and Guha, R.K. (2006) 'Understanding the intruder through attacks on cryptographic protocols', *Proceedings of the 44th ACM Southeast Conference* (*ACMSE2006*), March, pp.667–672.

Needham, R.M. and Schroeder, M. (1978) 'Using encryption for authentication in large networks of computers', *Communications of the ACM*, Vol. 21, No. 12, December, pp.993–999.

Oorschot, P.V. (1993) 'Extending cryptographic logics of belief to key agreement protocols', *Proceedings of the 1st ACM Conference on Computer and Communications Security*, November, pp.232–243.

Paulson, L.C. (1998) 'The inductive approach to verifying cryptographic protocols', *Journal of Computer Security*, Vol. 6, pp.85–128.

Ryan, P., Schneider, S., Goldsmith, M., Lowe, G. and Roscoe, B. (2000) *Modelling and Analysis of Security Protocols*, Addison-Wesley, Boston, MA, USA.

Shand, M. and Vuillemin, J. (1993) 'Fast implementations of RSA cryptography', *Proceedings of the 11th IEEE Symposium on Computer Arithmetic*, pp.252–259, Available online: http://ftp.digital.com/pub/DEC/PRL/researcharticles/SV93.ps.Z.

Song, D., Berezin, S. and Perrig, A. (2001) 'Athena: a novel approach to efficient automatic security protocol analysis', *Journal of Computer Security*, Vol. 9, pp.47–74.

Syverson, P. and Cervesato, I. (2001) 'The logic of authentication protocols', *Foundations of Security Analysis and Design*, Ser. LNCS 2171, Springer-Verlag, Berlin, Germany, pp.63–136.

Syverson, P. and Oorschot, P.V. (1994) 'On unifying some cryptographic protocol logics', *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, pp.14–28.

van Oorschot, P.C. (1993) 'Extending cryptographic logics of belief to key agreement protocols', *Proceedings of the 1st ACM Conference on Computer and Communications Security*, November, pp.233–243.

Winskel, G. (1987) 'Event structures', in Brauer, W., Reisig, W. and Rozenberg, G. (Eds.): *Petri Nets: Applications and Relationships to Other Models of Concurrency*, Ser. LNCS 255, Springer-Verlag, Berlin, Germany, pp.325–392.

Woo, T.Y.C. and Lam, S.S. (1992) 'Authentication for distributed systems', *Computer*, Vol. 25, No. 1, January, pp.39–52.

Notes

[1]Ideal cryptography means that a principal must have a key in order to perform any cryptographic operation (such as encryption, decryption, signatures) using that key.

[2]The word BAN was termed after its authors Burrows, Abadi and Needham. Since then, it has become tradition to call the extensions of BAN logic by the names of their authors, like GNY, VO and SVO.

[3]In BAN, a comma is used to represent concatenation of two messages X and Y, i.e., X, Y.

[4]In modal logic, these axioms are named K, T, 4, and 5 respectively and are known as axioms of the Lewis system S5 (Chellas, 1980).

[5]DTPL uses $\Rightarrow$ to represent the conditional instead of $\rightarrow$ used in the previous section in other logics. We follow the DTPL's notation in order to avoid confusion since DTPL uses $\rightarrow$ for other purposes.

[6]The DTPL uses to represent message concatenation. For simplicity, we write AB to represent the concatenation of the two terms A and B.

[7]In Caleiro *et al*. (2005a), the channel may output M to any possible aliases used by a principal.

[8]It is simply a matter of taste. Instead of using an entirely different notation for public keys (i.e., **PK**), we find our modified notation for public-keys $\mapsto$ to be closer to the corresponding notation for symmetric-key $\longleftrightarrow$. BAN also uses $\mapsto$ for public-keys.

[9]Most of the work in this field is based on this assumption. It simply says that a principal can not encrypt or decrypt a message without having the proper encryption or decryption key.

Reasoning about Belief in Cryptographic Protocols*

Li Gong, Roger Needham, and Raphael Yahalom†

University of Cambridge Computer Laboratory
Cambridge CB2 3QG, England

February 11, 1990

Abstract

Abstract. Analysis methods for cryptographic protocols have often focused on information leakage rather than on seeing whether a protocol meets its goals. Many protocols, however, fall far short of meeting their goals, sometimes for quite subtle reasons. We introduce a mechanism for reasoning about belief as a systematic way to understand the working of cryptographic protocols. Our mechanism captures more features of such protocols than that given in a recent work [1], to which our proposals are a substantial extension.

1 Introduction

Solutions to computer security problems over the last few years have brought forth the need for rigorous analysis methods. Formal tools must be provided to determine whether a solution indeed solves a problem, as well as to enable comparisons between proposed solutions. In this paper we propose a method for reasoning about cryptographic protocols in a distributed environment.

The work described was inspired by the recent development of a modal logic to reason about authentication protocols [1], which we refer to as the BAN logic. Indeed our work can be seen as a new approach within the framework proposed there. Like BAN, we aim to analyze a protocol step by step, make explicit any assumptions required, and draw conclusions about the final position it attains.

Our new approach seems to offer important advantages over the BAN approach. It does not require several universal assumptions which the BAN work does. For example, it does not assume that redundancy is always present in encrypted messages - incorporating instead a new notion of *recognizability* which captures a recipient's expectation of the contents of messages he receives. Also, it does not assume that a principal can always determine whether a message was not once originated by himself.

We distinguish between what one possesses and what one believes in. This allows us to treat separately the content of a message and the information *implied* by such a message. It also makes it possible to separate reasoning about the physical world from the reasoning about other principals' beliefs, so that we can consider different levels of trust in the reasoning.

The set of notions is expanded so that additional properties of messages can be incorporated in the reasoning process. Some of the existing notions are modified and made to correspond more naturally to execution states and are thus more intuitive. For example, encrypted and plaintext formulae are treated similarly, thus permitting reasoning about multiply encrypted messages. Also, plaintext messages, which are not considered in the BAN approach to be useful, can in some cases be used to derive further conclusions.

The new approach allows us to analyze a much wider range of protocols. A summary of the significant differences between our work and the BAN work appears at the end of the paper.

The rest of this paper is organized as follows. In the next section we outline the model of computation. In section 3 we introduce the notions and their corresponding notation. In section 4 we present selected logical postulates underlying our reasoning process. The complete set of postulates and their descriptions are included in appendix A. In section 5 our reasoning process is described. In section 6 we introduce reasoning about other principals' beliefs. In section 7 the Needham-Schroeder protocol is used as an example that highlights some of the characteristics of our approach. Finally, in section 8 we discuss our conclusions.

2 The Model of Computation

Our model of computation is similar to that used in the BAN work and possesses some characteristics of models used in other knowledge-theoretic work (e.g. [2]). We

*Published in Proceedings of the IEEE 1990 Symposium on Security and Privacy, Oakland, California, May, 1990, pp.234-248.

†Raphael Yahalom is supported by IBM's UK Academic Systems Marketing and by a British government's Overseas Research Students award.

outline the significant aspects of the model. A more formal presentation is given in appendix C.

A distributed environment consists of principals, essentially state-machines, which are connected by communication links. Messages on these links constitute the only means of communication between principals. Any principal can place a message on any link. He can also see and alter any message being passed along any link.

A protocol is a distributed algorithm. A protocol determines what messages should be sent by the participating principals as a function of their internal states. A run is a particular execution of the protocol. We refer to a protocol run as a session.

Each principal in each session maintains two sets: a *belief* set which includes all the current beliefs of the principal, and a *possession* set which includes all the formulae available to the principal. In particular, the latter includes everything the principal received, and everything the principal has generated himself, e.g. random numbers, in the current session.

Principals start a session with certain initial beliefs and initial possessions. From that point a principal can obtain new beliefs, and thus expand his belief set, as a result of receiving new messages. Inference rules, described below, enable the derivation of new beliefs from current beliefs and incoming messages. Similarly, a principal can increase his possessions.

Beliefs and possessions are monotonic within a given session. That is to say, if a belief or a possession is a member of its respective set at any phase of some session, then it is a member of that set at any subsequent phase of that session. However, no such claim is made across sessions. In particular, a belief or a possession of a principal in a session is not necessarily a member of the corresponding set in future, or past, sessions in which the principal participates.

The only universal assumption we require is that principals do not reveal their secrets. This is, in fact, only for convenience. We could let principals choose, as part of their initial beliefs, whether to trust each other in that respect. Without such basic trust, however, a principal would be rather limited in his ability to attain new beliefs.

3 Notions and Notation

In this section we introduce the basic notions underlying our reasoning process and their corresponding notation. The description below is informal and aims at providing intuitive understanding. The postulates presented in the next section, and the semantics in appendix C, provide a more precise definition.

3.1 Formulae

A formula is a name used to refer to a bit string, which would have a particular value in a run. This is rather like the name of a variable. Let X and Y range over formulae. Two kinds of special formulae, shared secrets and encryption keys, are denoted as S and K respectively. The following are also formulae:

(X, Y): conjunction of two formulae. We treat conjunctions as sets with properties such as associativity and commutativity.

$\{X\}_K$ and $\{X\}_K^{-1}$: conventional encryption and decryption (e.g. DES). It is assumed that the cryptosystems used are resistant to ciphertext-only and known-plaintext attacks. In addition, every bit in a ciphertext depends on all bits of the plaintext and the key in such a way that any change to the plaintext causes a random change in the ciphertext and vice versa. They satisfy $\{\{X\}_K\}_K^{-1} = X$, but not necessarily $\{\{X\}_K^{-1}\}_K = X$.

$\{X\}_{+K}$ and $\{X\}_{-K}$: public-key encryption and decryption. In addition to the requirements stated for conventional cryptosystems, they satisfy $\{\{X\}_{+K}\}_{-K} = X$. Some public-key schemes (e.g. RSA [8]) also satisfy that $\{\{X\}_{-K}\}_{+K} = X$.

$H(X)$: a one-way function of X. It is required that given X it is computationally feasible to compute $H(X)$; given $H(X)$ it is infeasible to compute X; it is infeasible to compute X and X' such that $X \neq X'$ but $H(X) = H(X')$ [5].

$F(X_1, \ldots, X_n)$: F is a many-to-one computationally feasible
function such that for any X_i, $1 \leq i \leq n$, and constants $C_1, \ldots, C_{n-1}$, $F(C_1, \ldots, C_{i-1}, X_i, C_i, \ldots, C_{n-1})$ is a one to one computationally feasible function, and its inverse is also computationally feasible. For example, exclusive-or is such a function. $F(X)$ denotes a computationally feasible one-to-one function whose inverse is also computationally feasible.

3.2 Statements

A basic statement reflects some property of a formula. Let P and Q range over principals. The following are basic statements.

$P \lhd X$: P *is told* formula X. P receives X, possibly after performing some computation such as decryption. That is, a formula being told can be the message itself, as well as any computable content of that message.

$P \ni X$: P *possesses*, or is capable of possessing, formula X. At a particular stage of a run, this includes all the formulae that P has been told, all the formulae he started the session with, and all the ones he has generated in that run. In addition P possesses, or is capable of possessing, everything that is computable from the formulae he already possesses.

$P \mid\sim X$: P *once conveyed* formula X. X can be a message itself or some content computable from such a message, i.e. a formula can be conveyed implicitly.

$P \mid\equiv \sharp(X)$: P *believes*, or is entitled to believe, that formula X is *fresh*. That is, X has not been used for the same purpose at any time before the current run of the protocol. For example, a counter or a random number generator (of sufficient quality) can serve to produce formulae that a principal believes to be fresh (called *nonces* [6]).

$P \mid\equiv \phi(X)$: P *believes*, or is entitled to believe, that formula X is *recognizable*. That is, P would recognize

X if P has certain expectations about the contents of X before actually receiving X. P may recognize a particular value (e.g. his own identifier), a particular structure (e.g. the format of a timestamp), or a particular form of redundancy.

$P \mid\equiv P \overset{S}{\leftrightarrow} Q$: P *believes*, or is entitled to believe, that S is a suitable *secret* for P and Q. They may properly use it to mutually prove identity. They may also use it as, or derive from it, a key to communicate. S will never be discovered by any principal except P, Q, or a principal trusted by either P or Q. However, in this case, the trusted principal should never use S as proof of identity or as a key to communicate. This notation is symmetrical: $Q \overset{S}{\leftrightarrow} P$ and $P \overset{S}{\leftrightarrow} Q$ can be used interchangeably.

$P \mid\equiv \overset{+K}{\mapsto} Q$: P *believes*, or is entitled to believe, that $+K$ is a suitable *public key* for Q. The matching secret key $-K$ will never be discovered by any principal except Q or a principal trusted by Q. In this case, however, the trusted principal should not use it to prove identity or to communicate.

Let C range over statements. The following are also statements:

C_1, C_2: conjunction. We treat conjunctions as sets with properties such as associativity and commutativity.

$P \mid\equiv C$: P believes, or P would be entitled to believe, that statement C holds.

3.3 "Not-Originated-Here" Formulae

A formula can be regarded as a *not-originated-here* formula, denoted by prefixing a star to the formula, e.g. $*X$. Statement $P \lhd *X$ indicates that P is told a formula which he did not convey previously in the current run.

Suppose a shared secret key K is used between P and Q. If P constructs and conveys a formula $\{X\}_K$ and P is later told the exact packet, possibly encrypted under other keys, then P should not reason that Q once conveyed X. The reason is that P could have sent it himself. A not-originated-here formula implies that it was not first conveyed by the recipient in this session (but could have been conveyed by him in previous sessions). A principal can believe he has never conveyed a formula that he receives if it is a not-originated-here formula and he also believes the formula to be fresh.

We use a pattern scanner to screen a normal protocol description and insert those stars at the correct places. The algorithm is discussed in section 5.

In appendix B we discuss reasoning in environments where principals may believe that they can identify messages that were not originated by themselves in any session.

4 Logical Postulates

In this section we introduce the logical postulates underlying the reasoning process. There are five categories of postulates. We describe each category and present representative postulates. A complete list of all the logical postulates and their description is included in appendix A.

4.1 Being-Told Rules

The first set of rules deals with formulae a principal receives. We regard every formula a principal receives, as well as certain manipulations of that formula (e.g. decryption with certain keys), as *being told* to that principal. The following are examples of *being-told* rules:

T2 $$\frac{P \lhd (X, Y)}{P \lhd X}$$

Being told a formula implies being told each of its concatenated components.

T3 $$\frac{P \lhd \{X\}_K,\ P \ni \mathrm{K}}{P \lhd X}$$

If a principal is told a formula encrypted with a key he possesses then he is also considered to have been told the decrypted contents of that formula.

4.2 Possession Rules

The next set of rules specify the formulae a principal is capable of possessing by manipulating formulae he already possesses. Examples of such rules are:

P1 $$\frac{P \lhd X}{P \ni X}$$

A principal is capable of possessing anything he is told.

P2 $$\frac{P \ni X,\ P \ni Y}{P \ni (X, Y),\ P \ni F(X, Y)}$$

If a principal possesses two formulae then he is capable of possessing the concatenation of the two formulae and a function F of them.

4.3 Freshness Rules

Freshness rules specify the formulae a principal can believe to be fresh, given his beliefs about the freshness of other formulae. Recall that a principal's belief in the freshness of a formula represents his belief that the formula has never had the same value in any previous run of the protocol.

F1 $$\frac{P \mid\equiv \sharp(X)}{P \mid\equiv \sharp(X, Y),\ P \mid\equiv \sharp(F(X))}$$

If P believes a formula X is fresh, then he is entitled to believe that any formula of which X is a component is fresh, and that a computationally feasible one-to-one function F of X is fresh. For convenience, we use $P \mid\equiv \sharp(X, Y)$ to denote $P \mid\equiv \sharp(X)$ or $P \mid\equiv \sharp(Y)$.

F2 $$\frac{P \mid\equiv \sharp(X),\ P \ni K}{P \mid\equiv \sharp(\{X\}_K),\ P \mid\equiv \sharp(\{X\}^{-1}_K)}$$

If P believes a formula X is fresh and possesses a key, then P is entitled to believe that the encryption, as well as the decryption, of X with that key is fresh.

4.4 Recognizability Rules

Recognizability rules specify the formulae a principal can believe to be recognizable, given his beliefs about the recognizability of other formulae. Examples of recognizability rules are:

R1 $$\frac{P \mid\equiv \phi(X)}{P \mid\equiv \phi(X,Y),\ P \mid\equiv \phi(F(X))}$$

If a principal P believes a formula X is recognizable, then he is entitled to believe that any formula of which X is a component is recognizable, and that a computationally feasible function F of X is recognizable.

R2 $$\frac{P \mid\equiv \phi(X),\ P \ni K}{P \mid\equiv \phi(\{X\}_K),\ P \mid\equiv \phi(\{X\}^{-1}_K)}$$

If a principal P believes a formula X is recognizable, and P possesses a key K, then P is entitled to believe that the encryption and the decryption of X with K are recognizable.

4.5 Message Interpretation Rules

Finally, we have rules which enable principals to advance their beliefs about other principals by examining messages they receive.

Postulate I1 below (as well as I2 and I3 in appendix A) includes a freshness requirement that may seem rather surprising. It is a consequence of the fact that when a key is shared between two (or more) principals, each could have used the secret to construct a formula. A principal should only be convinced that a message based on a shared secret is not a replay of one of his own messages if the message has a not-originated-here star associated with it, and he believes the message is fresh.

I1 $$\frac{P \triangleleft *\{X\}_K,\ P \ni K,\ P \mid\equiv P \overset{K}{\leftrightarrow} Q,\ P \mid\equiv \phi(X),\ P \mid\equiv \sharp(X,K)}{P \mid\equiv Q \mid\sim X,\ P \mid\equiv Q \mid\sim \{X\}_K,\ P \mid\equiv Q \ni K}$$

If for a principal P, all of the following conditions hold: (1) P receives a formula consisting of X encrypted with K and marked with a not-originated-here mark; (2) P possesses key K; (3) P believes K is a suitable secret for himself and Q; (4) P believes formula X is recognizable; (5) P believes that K or X are fresh. Then P is entitled to believe the following: Q once conveyed X; Q once conveyed the formula X encrypted with K; and Q possesses K.

I6 is an example of rules that permit reasoning about the state of a sender:

I6 $$\frac{P \mid\equiv Q \mid\sim X,\ P \mid\equiv \sharp(X)}{P \mid\equiv Q \ni X}$$

If P believes that Q once conveyed formula X and P believes that X is fresh, then P is entitled to believe that Q possesses X.

4.6 Rationality Rule

We supplement the postulates with a rule that we call the rationality rule. Informally, it states that our set of postulates can be expanded to permit reasoning about a principal's beliefs regarding the state of other principals. More precisely:

if $\frac{C1}{C2}$ is a postulate, then for any principal P, so is $\frac{P \mid\equiv C1}{P \mid\equiv C2}$.

For example, from postulate P2 we can obtain the following postulate:

$$\frac{Q \mid\equiv P \ni X,\ Q \mid\equiv P \ni Y}{Q \mid\equiv P \ni F(X,Y)}$$

If Q believes that P is capable of possessing two formulae then Q believes that P is capable of possessing the concatenation of the two formulae and a function F of them.

The rationality rule represents our view that principals are capable of deriving rational conclusions about the state of other principals. Issues associated with reasoning about other principals' beliefs are discussed in section 6.

5 Protocol Analysis

5.1 A Protocol Parser

Protocols are typically described by listing messages sent between the principals, and by symbolically showing the source, the destination, and the contents of each message. This form is intuitively easy to understand. We require only simple transformations to attain a form suitable for direct manipulation in our logic.

The main point is related to the fact that a typical protocol description does not make a distinction between X and $*X$. $P \triangleleft *X$ denotes the fact that a P is not the first one to convey X in the current run of the protocol, a fact that is only implicitly included in the description itself. We design a parser that explicitly inserts the stars to a protocol description, thus avoiding a much more complex form of logic that would have otherwise been required. We sketch out a general description of the parser algorithm.

For each line from the description $P \to Q : X$, if $Q = P$ an error is reported; otherwise, the parser produces two lines, $P \mid\sim X$ followed by Q $\triangleleft X$. For each principal P, the parser examines all lines of the form $P \mid\sim X$ or $P \triangleleft X$. It then scans from the beginning. For each pattern of a complete formula Y in a line $P \lhd X$, if Y does not first appear in a line $P \mid\sim X$ in the protocol, the parser inserts a star before Y. The parser would also mark $(*X, *Y)$ instead of $*(X, Y)$. After these are completed, the parser drops all lines of the form $P \mid\sim X$.

In addition, for private-key encryption, the parser replaces patterns of the form $\{\{X\}_K\}_K^{-1}$ with X. Similar replacements are performed for public-key encryption.

5.2 Annotated Assertions

The protocol analysis consists of annotating the protocols with statements and manipulating these statements with the postulates. A protocol is a sequence of *told*-statements $C_1, \cdots, C_n$, each of the form $P \lhd X$. An annotation for a protocol consists of a sequence of assertions, conjunctions of statements, inserted before the first told-statement and after each told-statement. The first assertion contains the assumptions and the last contains the conclusions. They can be understood as formulae in Hoare logic [3]. As in BAN [1], if the assumptions hold, each assertion should hold after the execution of its respective protocol prefix. The assertions are derived by the syntactic application of the above postulates to statements. Often, the goal of an analysis is to derive the final positions of each of the principals at the end of a given protocol. That final position is represented by the corresponding assertions.

5.3 Example - A Voting Protocol

We present a simple voting protocol, outline its characteristics, and demonstrate the reasoning process as described thus far.

The environment consists of a coordinator Q and n participants P_i. Communication is performed by message passing. Each participant P_i shares a secret with Q, denoted as S_i. The goal of the protocol is to elect a principal for some position. Each participant has one vote V_i. The coordinator determines the winner according to certain rules.

The protocol requirements specify that the coordinator should be able to identify votes for the current election and that each principal who possesses the result R at the end of the protocol should be convinced that it was generated by the coordinator during the current run.

The protocol consists of three messages between the coordinator and each participant. These correspond to request-vote, voting, and result-announcement phases. N_q, and N_i are nonces generated by Q and P_i respectively. A secret S used for identification purposes is denoted $< S >$ (see appendix A.5).

1. $Q \rightarrow P_i$: N_q
2. $P_i \rightarrow Q$: $P_i, N_i, V_i, H(N_q, < S_i >, V_i)$
3. $Q \rightarrow P_i$: $R, H(N_i, < S_i >, R)$

The parser algorithm would produce the following description of the protocol.

1. $P_i \lhd$: $*N_q$
2. $Q \lhd$: $*P_i, *N_i, *V_i, *H(N_q, * < S_i >, V_i)$
3. $P_i \lhd$: $*R, *H(N_i, < S_i >, R)$

5.4 Protocol Analysis

We assume that the following holds at the beginning of every run of the protocol:

$$P_i \ni S_i; \quad P_i \ni N_i; \quad P_i \mid\equiv Q \overset{S_i}{\leftrightarrow} P_i; \quad P_i \mid\equiv \sharp(N_i)$$

$$Q \ni S_i; \quad Q \ni N_q; \quad Q \mid\equiv Q \overset{S_i}{\leftrightarrow} P_i; \quad Q \mid\equiv \sharp(N_q)$$

That is, each one of the participants possesses a secret and believes that it is for himself and the coordinator. Each also possesses a nonce and believes in its freshness. Similarly the coordinator possesses the secrets and believes each for him and a principal. He also possesses a nonce and believes it is fresh.

For any run of the protocol:

Message 1: Applying T1 and P1 we obtain $P_i \ni N_q$. That is P_i possesses N_q.

Message 2: Applying T1, T2, and P1 we obtain $Q \ni (V_i, N_i)$. Q possesses V_i and N_i.

Applying F1 we obtain $Q \mid\equiv \sharp(V_i, N_q, S_i)$. Q believes that (V_i, N_q, S_i) is fresh. That is, it is generated during the current run of the protocol and so cannot be a replay of a message from previous runs.

Applying I3 and I7 we obtain $Q \mid\equiv P_i \mid\sim (V_i, N_q)$. Applying F1 we obtain $Q \mid\equiv \sharp(V_i, N_q)$. Q believes that P_i conveyed (V_i, N_q) in the current run.

Similarly, in message 3, P_i believes that Q conveyed R in the current run.

The final position that the coordinator and each of the participant attain correspond to the specification goals outlined earlier. In particular, the coordinator is in a position to recognize fresh and valid vote messages from all participants and to properly determine the winner. The participants possess a result which was generated by the coordinator in the current session.

The reasoning process based on the postulates above can be considered a *physical level* reasoning process. Each principal can only advance his beliefs and increase his possessions based on the physical content of the messages he receives. Consequently a principal can attain beliefs about who sent what and when, but not about the sender's beliefs at the time the message was sent. In the voting protocol the participants are convinced that the result was originated by the coordinator, but there is no way of deriving conclusions such as that the coordinator himself believes in the validity of the result. Thus far, our formal protocol description did not include such notions. They are discussed in the next section.

6 Beliefs about Others' Beliefs

The reasoning process described thus far permits conclusions of the following kind: $P \mid\equiv Q \ni X$, $P \mid\equiv Q \mid\sim$

X, $P \mid\equiv \sharp(X)$, and so on. These are principal P's beliefs about the physical world. However, it does not allow for conclusions such as $P \mid\equiv Q \mid\equiv \sharp(X)$, i.e. P cannot draw any conclusion about beliefs held by other principals.

We choose to separate these two types of beliefs for several reasons. First, by examining the contents of messages one can, quite directly, derive conclusions about the possessor of a formula or the conveyor of a message, that is, reason about the physical world. At that level there is no need for reasoning about other principals' beliefs.

Secondly, since each principal expresses his beliefs by sending messages, we choose to interpret the beliefs on the sender's part as the preconditions for a message to be sent at all. That is, the recipient should interpret it as such when receiving the message. A message of the same form may carry different, context dependent, meanings in different protocols. Having believed a message to be genuine, a recipient can choose to believe the sender's beliefs, if he trusts the sender's honesty and competence in following the protocol specification. In other words, since we do not require the universal assumption that all principals are honest and competent, we should reason about beliefs held by others based on trust of different levels.

Thirdly, typical protocol specifications often include verbal description to the effect that a principal should proceed only if certain conditions hold or only if he holds certain beliefs. This can be regarded as a precondition. Thus we feel that our approach is natural.

The precondition of a formula X being conveyed, represented by statement C, is described in the protocol as $X \leadsto C$. We call C a *message extension*. Since a message can contain a number of formulae that are destined for different principals and possibly with different meanings, we use $P \triangleleft (X_1 \leadsto C_1), (X_2 \leadsto C_2)$ to clearly express the scope.

Now recall that the parser may find a pattern that first appears in a line of the form $P \mid\sim X$. This may indicate that a new formula is conveyed by P under a certain precondition. We shall take, for example, the verbal explanation often found in protocol descriptions, translate this into the language of our logic, and insert it after the formula. The same condition should be replicated whenever the same pattern reappears in the following messages. As before, only lines of the form $P \triangleleft (X \leadsto C)$ are kept. For simplicity, a formula without a star prefix has no conditions attached, since no principal should derive new conclusions from such a formula. The extension of a message is considered part of that message, and so the conclusion that a principal once conveyed a formula can be augmented to the conclusion that he conveyed the formula and its extension - if such an extension is associated with the formula.

6.1 Trust and Jurisdiction

We use the notion of *jurisdiction* to represent trust and delegation.

$P \mid\equiv Q \mid\Longrightarrow C$: P believes that Q has jurisdiction over statement C. That is, P believes that principal Q is an authority on C and should be trusted on this matter.

The following postulate describes jurisdiction more formally:

$$\textbf{J1} \qquad \frac{P \mid\equiv Q \mid\Longrightarrow C,\ P \mid\equiv Q \mid\equiv C}{P \mid\equiv C}$$

J1 states that if P believes that Q has jurisdiction over some statement C and that Q believes in C, then P ought to believe in C as well. The following construct is a special case of jurisdiction.

$P \mid\equiv Q \mid\Longrightarrow Q \mid\equiv *$: P believes that Q has jurisdiction over all his beliefs. The principal Q is considered by P to be completely honest and competent.

J2 states that if P believes that Q is honest and competent, and P receives a message $X \leadsto C$, which is originated from Q, then P ought to believe that Q really believes C. J3 is a special case.

$$\textbf{J2} \qquad \frac{\begin{array}{c}P \mid\equiv Q \mid\Longrightarrow Q \mid\equiv *,\ P \mid\equiv Q \mid\sim (X \leadsto C),\\ P \mid\equiv \sharp(X)\end{array}}{P \mid\equiv Q \mid\equiv C}$$

$$\textbf{J3} \qquad \frac{P \mid\equiv Q \mid\Longrightarrow Q \mid\equiv *,\ P \mid\equiv Q \mid\equiv Q \mid\equiv C}{P \mid\equiv Q \mid\equiv C}$$

Other levels of trust could be defined and used for reasoning. For example, one could introduce a notion representing the fact that a principal is following the protocol. In other words, the relationship between the contents of the messages he sends and their *meaning* is maintained. Consequently, a similar postulate to J2, based on that notion, would conclude $P \mid\equiv Q \mid\sim Q \mid\equiv C$ rather than $P \mid\equiv Q \mid\equiv C$. Clearly, this represents a lower level of trust and so enables weaker conclusions to be derived. Moreover, any of the above levels of trust could be applied to any aspect of a principal's behavior. This reflects the fact that principals are often trusted differently with respect to the different tasks they perform.

Recall that the rationality rule introduced in section 4 enables principals to further increase their beliefs concerning other principals' beliefs.

6.2 Consistency of Protocol Description

Our approach to reasoning about principals' states enables us to detect invalid or inconsistent protocol descriptions. In particular, a principal should include in a message only formulae he possesses at the time the message is constructed. Similarly, a principal should imply only his beliefs which, according to our reasoning, he holds at the time the message is sent. Note that present discussion refers to the protocol description - during protocol execution principals may lie.

The following two checks are thus performed: *possession consistency*, i.e. a principal should only be able to include in any message he sends, a formula he possesses; *belief consistency*, i.e. a message extension should include only beliefs held by the sender at the time the message is sent.

7 The Needham-Schroeder Protocol

In this section we apply the reasoning process to the Needham-Schroeder authentication protocol [6]. We choose this protocol as an example because it influenced the design of a significant number of existing systems and published protocols, and because it serves as one of the examples in the BAN logic paper [1]. Our analysis below can thus provide insight to some of the differences between the approaches, and to the advantages that ours offers. In the rest of this section we describe the protocol, analyze it, and finally investigate how modifications to the protocol affect the final position of the participants.

7.1 Protocol Description

The general goal of the protocol is for two principals P and Q to be provided with a shared secret [6, 1]. That secret can consequently be used as a session key. There exists a trusted *authentication server* S, which shares common secrets with all potential participants and can generate good quality sessions keys.

Different authentication protocols may differ in the final positions which the principals attain. Different positions seem to suffice for different environments. It is usually assumed that the minimum requirement for an end of a run is that each principal would possess a session key, and be convinced of the validity and quality of that key. Often, in addition, each principal is required to believe something about the state of the other principal. That can vary between believing that the other principal is operational, that the other principal possesses the key, or that the other principal believes in the validity of the possessed key.

The Needham-Schroeder protocol consists of the following five messages:

1. $P \rightarrow S$: P, Q, N_p
2. $S \rightarrow P$: $\{N_p, Q, K, \{K, P\}_{K_{qs}}\}_{K_{ps}}$
3. $P \rightarrow Q$: $\{K, P\}_{K_{qs}}$
4. $Q \rightarrow P$: $\{N_q\}_K$
5. $P \rightarrow Q$: $\{N_q - 1\}_K$

N_p and N_q are nonces for P and Q respectively, K_{ps} and K_{qs} are the secrets between P and S, Q and S respectively. K is the session key for P and Q generated by S. To the above description, we add message extensions which reflects the verbal explanation of the protocol execution. The validity of these extensions will be discussed during the analysis of the protocol. We use F to denote the decrement computation. The parser would produce the following output.

1. $S \lhd: *P, *Q, *N_p$
2. $P \lhd: *\{N_p, Q, *K, *\{K, P\}_{K_{qs}} \leadsto S \mid\equiv P \overset{K}{\leftrightarrow} Q\}_{K_{ps}} \leadsto S \mid\equiv P \overset{K}{\leftrightarrow} Q$
3. $Q \lhd: *\{*K, *P\}_{K_{qs}} \leadsto S \mid\equiv P \overset{K}{\leftrightarrow} Q$
4. $P \lhd: *\{*N_q\}_K$
5. $Q \lhd: *\{*F(N_q)\}_K \leadsto P \mid\equiv P \overset{K}{\leftrightarrow} Q$

7.2 Protocol Analysis

We begin by listing the initial assumptions:

$P \ni K_{ps}$; $P \ni N_p$

$P \mid\equiv P \overset{K_{ps}}{\leftrightarrow} S$; $P \mid\equiv \sharp(N_p)$; $P \mid\equiv \phi(Q)$

$Q \ni K_{qs}$; $Q \ni N_q$

$Q \mid\equiv Q \overset{K_{qs}}{\leftrightarrow} S$; $Q \mid\equiv \sharp(N_q)$; $Q \mid\equiv \phi(N_q)$

That is, each principal possesses a secret and believes it is a secret between himself and the authentication server. He also possesses a nonce which he believes to be fresh. In addition, P believes that the identifier Q is recognizable and Q believes that N_q is recognizable.

$P \mid\equiv S \Longrightarrow (P \overset{K}{\leftrightarrow} Q)$; $P \mid\equiv S \Longrightarrow S \mid\equiv *$;
$P \mid\equiv Q \Longrightarrow Q \mid\equiv *$

$Q \mid\equiv S \Longrightarrow (P \overset{K}{\leftrightarrow} Q)$; $Q \mid\equiv S \Longrightarrow S \mid\equiv *$;
$Q \mid\equiv P \Longrightarrow P \mid\equiv *$

P and Q believe in the jurisdiction of S over quality secrets to be shared between them. P and Q also believe that S is honest and competent. Moreover, P believes Q is competent and honest, and vice versa. These two assumptions are not essential. The protocol could attain a useful final position even if the principals do not completely trust each other to tell the truth. However, as we shall see, the added trust represented by these last two assumptions enables both P and Q to attain stronger final positions.

$S \ni K_{ps}$; $S \ni K_{qs}$; $S \ni K$

$S \mid\equiv P \overset{K_{ps}}{\leftrightarrow} S$; $S \mid\equiv Q \overset{K_{qs}}{\leftrightarrow} S$; $S \mid\equiv P \overset{K}{\leftrightarrow} Q$

S believes that he possesses valid keys with P and Q. He also believes that K is a suitable secret for P and Q.

For any run of the protocol:

Message 1: applying T1 and P1 we obtain $S \ni (P, Q, N_p)$. That is S possesses P, Q, and N_p.

Message 2: first we note that the extension to the message, $S \mid\equiv P \overset{K}{\leftrightarrow} Q$, is valid because it holds when the message is sent as is evident from the initial assumptions. Also S is also sure that the recipient, P, cannot mistake K as a key for himself and a principal other than Q, since the name Q is included in the message.

Applying T1, T3, and P1 we get $P \ni (N_p, Q, K, \{K, P\}_{K_{qs}})$. P possesses the message contents.

Applying T2 we obtain $P \ni K$. P possesses the new key K.

Applying F1 we obtain $P \mid\equiv \sharp(N_p, Q, K, \{K, P\}_{K_{qs}})$. P believes the message is not a replay.

Applying R1 we obtain $P \mid\equiv \phi(N_p, Q, K, \{K, P\}_{K_{qs}})$. P believes the contents of the message are recognizable.

Applying I1 we obtain $P \mid\equiv S \mid\sim (N_p, Q, K, \{K, P\}_{K_{qs}})$. P believes the message originated from S.

Applying J2 we obtain $P \mid\equiv S \mid\equiv P \overset{K}{\leftrightarrow} Q$. P believes S believes that K is a good key for P and Q.

Applying J1 we obtain $P \mid\equiv P \overset{K}{\leftrightarrow} Q$. P believes that K is a suitable key for P and Q.

Message 3: the extension to the message, $S \mid\equiv P \overset{K}{\leftrightarrow} Q$, is valid.

Applying T1, T3 and P1 we obtain $Q \ni K$. Q possesses K.

None of our postulates enable us to further derive new useful beliefs or possessions from this message. In particular, we cannot derive the freshness of the message. Indeed, as far as Q is concerned it could very well be a replay of a message from previous runs. Unlike the BAN logic, our rules cannot even provide the conclusion that S has *once* sent the message. Q has no reason to know this is not a replay of a message he once sent himself, unless a principal can identify messages that were not originated by themselves. Finally, because Q is not sure that S conveyed the message at all, or that it is fresh, Q obviously can not convince himself about S's current beliefs and cannot make use of the extension to the message.

Message 4: applying T1, T3 and P1 we obtain $P \ni N_q$. P possesses N_q.

No further conclusions can be derived. Although K is used, P does not know who the originator of the message is, or that Q now possesses the key. The reason is that N_q, a random number generated by Q, is not recognizable to P. Thus there is not sufficient redundancy for P to be convinced of the genuineness of the decrypted content.

Message 5: none of our postulates can derive any useful belief or possession from that message. In particular Q cannot even be convinced that it was sent by P, much less draw any conclusions on P's state. Although the contents are accessible to Q who possess K, and the contents are recognizable to Q (postulate R1), Q is not convinced that he shares the key with P.

If we include a notion of *somebody else*, then we can derive that Q believes that somebody (not himself) who possesses K sent the message.

To conclude, our reasoning leads us to the conclusions that given the first three messages of the protocol, the last two messages attain nothing of use and can thus be eliminated without weakening the final position. However some modifications enable the derivation of the much improved final position that was originally intended by the protocol authors.

7.3 The Enhanced Needham-Schroeder Protocol

Recently, Needham and Schroeder suggested the following modification to their original protocol [7]:

1. $P \rightarrow Q$: P
2. $Q \rightarrow P$: $\{P, N_{q1}\}_{K_{qs}}$
3. $P \rightarrow S$: $P, Q, N_p, \{P, N_{q1}\}_{K_{qs}}$
4. $S \rightarrow P$: $\{N_p, Q, K, \{K, N_{q1}, P\}_{K_{qs}}\}_{K_{ps}}$
5. $P \rightarrow Q$: $\{K, N_{q1}, P\}_{K_{qs}}$
6. $Q \rightarrow P$: $\{N_q\}_K$
7. $P \rightarrow Q$: $\{N_q - 1\}_K$

Q believes his nonce N_{q1} to be fresh. The difference between the two versions is that the enhanced version begins with an exchange between P and Q. P can thus provide Q's nonce to the server, and S includes that nonce in his response, which P forwards in message 5.

Similar reasoning to that of P, after message 2 in the original protocol, leads us to derive that after message 5 we obtain $Q \mid\equiv P \overset{K}{\leftrightarrow} Q$. Q believes that K is a suitable key for P and Q. That enables us to add the above belief as an extension to message 6:

$P \lhd *\{*N_q\}_K \leadsto Q \mid\equiv P \overset{K}{\leftrightarrow} Q$

However, P cannot deduce any conclusions from that message as it does not contain anything recognizable to P, as noted in the previous section. P does however gain the possession of N_q (postulates T1, T3 and P1).

Message 7: this message in the modified protocol is identical to message 5 in the original one:

$Q \lhd *\{*F(N_q)\}_K \leadsto P \mid\equiv P \overset{K}{\leftrightarrow} Q$

As usual, we first check that the extension is valid. Indeed it is, as we concluded that P holds that belief when he sends message 7.

Applying F1 and F2 we obtain $Q \mid\equiv \sharp(\{F(N_q)\}_K)$

Applying R1 we obtain $Q \mid\equiv \phi(F(N_q))$.

Applying I1 we obtain $Q \mid\equiv P \mid\sim (\{F(N_q)\}_K)$

Applying J2 we obtain $Q \mid\equiv P \mid\equiv P \overset{K}{\leftrightarrow} Q$

7.4 A Further Modification

The modification required to attain the desired final position is to include a formula in message 6 that would be recognizable to P. We propose the following modified version of message 6:

6. $P \lhd *\{*N_q, Q\}_K \leadsto Q \mid\equiv P \overset{K}{\leftrightarrow} Q$

Recalling that P believes the identifier Q to be recognizable (initial assumptions), similar reasoning to that described above would lead us to conclude:

$P \mid\equiv Q \mid\equiv P \overset{K}{\leftrightarrow} Q$

Or, if P does not trust Q's competence or honesty,

$P \mid\equiv Q \ni K$

7.5 Analysis Summary

We summarize the difference in the final positions that the above protocols attain. The original protocol attains a rather weak position:

$P \ni K; \quad P \mid\equiv P \overset{K}{\leftrightarrow} Q$

$Q \ni K$

That is, P possesses and believes a mutual secret, Q possesses the secret but cannot believe it. Neither believe anything about the other.

The extended protocol attains a stronger position even if P and Q do not trust each other's competence or honesty.

$P \ni K; \quad P \mid\equiv P \overset{K}{\leftrightarrow} Q$

$Q \ni K; \quad Q \mid\equiv P \overset{K}{\leftrightarrow} Q; \quad Q \mid\equiv P \ni K$

That is, P and Q possess the key and believe in it and Q believes that P possesses it. If in addition Q trusts P's competence and honesty, our reasoning process allows us to conclude that the extended protocol attains:

$P \ni K; \quad P \mid\equiv P \overset{K}{\leftrightarrow} Q$

$Q \ni K; \quad Q \mid\equiv P \overset{K}{\leftrightarrow} Q; \quad Q \mid\equiv P \ni K;$

$\quad Q \mid\equiv P \mid\equiv P \overset{K}{\leftrightarrow} Q$

That is, P and Q possess the key and believe in it, and Q believes that P possesses it and believes in it. Finally, our modified version attains:

$P \ni K; \quad P \mid\equiv P \overset{K}{\leftrightarrow} Q; \quad P \mid\equiv Q \ni K;$

$\quad P \mid\equiv Q \mid\equiv P \overset{K}{\leftrightarrow} Q$

$Q \ni K; \quad Q \mid\equiv P \overset{K}{\leftrightarrow} Q; \quad Q \mid\equiv P \ni K;$

$\quad Q \mid\equiv P \mid\equiv P \overset{K}{\leftrightarrow} Q$

That is, P and Q possess the key and believe in it and each believes that the other possesses it and believes in it, assuming they trust each other. We note that the Yahalom protocol [1] can attain the same final position with five rather than seven messages.

8 Conclusions

We presented a new approach to reasoning about cryptographic protocols. Our work, which was inspired by the BAN logic work, offers significant advantages. Some of the main differences between the two approaches are summarized below.

The notion of *possession* incorporated in our approach assumes that principals can include in messages data they do not *believe* in, but merely possess. This also enables us to derive conclusions such as "Q possesses the shared key" in the example in the previous section. The BAN approach, on the other hand, requires all principals to include in messages only formulae in which they believe, and only allows the derivation of stronger conclusions of the kind "Q believes in the shared key", if they hold.

Our approach places a strong emphasis on the separation between the content and the meaning of messages. This can increase consistency in the analysis and, more importantly, introduce the ability to reason at more than one level. The final position in a given run will depend on the level of mutual trust of the specific principals participating in that run.

The notion of *recognizability* is an important one. With this notion, it is possible to express the ability of a recipient to identify the messages he expects. The BAN logic assumes that the encryption component always provides enough redundancy, which may not always be necessary. In fact, it is sometimes rather undesirable [4].

The *not-originated-here* notion allows us to determine that certain messages are not replays of a recipient's own previous messages in a session. BAN assumes that the encryption always provides information that identifies the sender. This can be achieved by, for example, embedding a sender's unique (and perhaps permanent) identifier in every message. We believe that such a property should not always be required, and in any case needs to be specified explicitly at the protocol description level.

The many additional constructs and rules we have introduced extend the scope of the protocols that can be analyzed. The new approach requires less assumptions and can thus be seen as more general. Also, it is not limited to authentication protocols, and can be used to analyze, for example, some cryptographic protocols that make use of one-way functions.

Like BAN, we choose not to include elaborate temporal notions or any form of negation. This helps to simplify the analysis process, while being sufficient for the derivation of important characteristics of cryptographic protocols. Extending that model is a subject of current investigation.

Finally we note that beliefs are not associated with any guarantee of truth with respect to the real world state they represent. It seems that beliefs are an appropriate notion to use in an environment of potential mutual suspicion such as the one we assume.

9 Acknowledgments

We would like to thank Martin Abadi and Mike Burrows of DEC Systems Research Center, Bill Harbison, Mark Lomas, and David Wheeler of the University of Cambridge Computer Laboratory, and the referees for valuable comments. Susanna Wills helped to improve the presentation.

References

[1] M. Burrows, M. Abadi, and R. Needham, "A Logic of Authentication", in Proceedings of the 12th ACM Symposium on Operating Systems Principles, Litchfield Park, Arizona, December, 1989. Published as *ACM Operating System Review*, Vol.23, No.5, pp.1-13, December, 1989. A fuller version was published as DEC System Research Center Report No.39, Palo Alto, California, February, 1989.

[2] J.Y. Halpern and Y. Moses, "Knowledge and Common Knowledge in a Distributed Environment", in Proceedings of the 3rd ACM Symposium on Principles of Distributed Computing, pp.50-61, Vancouver, British Columbia, August, 1984.

[3] C.A.R. Hoare, “An Axiomatic Basis for Computer Programming”, *Communications of the ACM*, Vol.12, No.10, pp.576-580 and p.583, October, 1969.

[4] T.M.A. Lomas, L. Gong, J.H. Saltzer, and R.M. Needham, “Reducing Risks from Poorly Chosen Keys”, in Proceedings of the 12th ACM Symposium on Operating Systems Principles, Litchfield Park, Arizona, December, 1989. Published as *ACM Operating System Review*, Vol.23, No.5, pp.14-18, December, 1989.

[5] R.C. Merkle, “One Way Hash Functions and DES”, in *Advances of Cryptology*, Proceedings of Crypto ’89, Santa Barbara, California, October, 1989.

[6] R.M. Needham and M.D. Schroeder, “Using Encryption For Authentication in Large Networks of Computers”, *Communications of the ACM*, Vol.21, No.12, pp.993-999, December, 1978.

[7] R.M. Needham and M.D. Schroeder, “Authentication Revisited”, *Operating Systems Review*, Vol.21, No.1, p.7, January, 1987.

[8] R.L. Rivest, A. Shamir, and L. Adleman, “A Method for Obtaining Digital Signatures and Public-key Cryptosystems”, *Communications of the ACM*, Vol.21, No.2, pp.120-126, February, 1978.

A Logical Postulates

In this appendix we list all the logical postulates and their description. A postulate that applies to formula X also applies to $*X$, though not necessarily vice versa.

We recall that these postulates can be supplemented with the following rationality rule. Informally, it states that our set of postulates can be expanded to permit reasoning about a principal's beliefs regarding the state of other principals. More precisely:

if $\frac{C1}{C2}$ is a postulate, then for any principal P, so is $\frac{P \mid\equiv C1}{P \mid\equiv C2}$.

A.1 Being-Told Rules

T1 $$\frac{P \lhd *X}{P \lhd X}$$

Being told a “not-originated-here” formula is a special case of being told a formula.

T2 $$\frac{P \lhd (X, Y)}{P \lhd X}$$

Being told a formula implies being told each of its concatenated components.

T3 $$\frac{P \lhd \{X\}_K,\ P \ni K}{P \lhd X}$$

If a principal is told a formula encrypted with a key he possesses then he is considered to have also been told the decrypted contents of that formula.

T4 $$\frac{P \lhd \{X\}_{+K},\ P \ni -K}{P \lhd X}$$

If a principal is told a formula encrypted with a public key and he possesses the corresponding private key then he is considered to have also been told the decrypted contents of that formula.

T5 $$\frac{P \lhd F(X, Y),\ P \ni X}{P \lhd Y}$$

If a principal is told the result of a function F, and if he possesses one of the two arguments, then he is considered to have been told the other argument as well (see definition of F in section 3.1).

For RSA, or any public-key system with the property $\{\{X\}_{-K}\}_{+K} = X$, the following postulate holds:

T6 $$\frac{P \lhd \{X\}_{-K},\ P \ni +K}{P \lhd X}$$

If a principal is told a formula encrypted with a private key and he possesses the corresponding public key then he is considered to have also been told the decrypted contents of that formula.

A.2 Possession Rules

P1 $$\frac{P \lhd X}{P \ni X}$$

A principal is capable of possessing anything he is told.

P2 $$\frac{P \ni X,\ P \ni Y}{P \ni (X, Y),\ P \ni F(X, Y)}$$

If a principal possesses two formulae then he is capable of possessing the formula constructed by concatenating the two formulae, as well as a computationally feasible function F of them.

P3 $$\frac{P \ni (X, Y)}{P \ni X}$$

If a principal possesses a formula then he is capable of possessing any one of the concatenated components of that formula.

P4 $$\frac{P \ni X}{P \ni H(X)}$$

If a principal possesses a formula then he is capable of possessing a one-way computationally feasible function of that formula.

P5 $$\frac{P \ni F(X,Y),\ P \ni X}{P \ni Y}$$

If a principal possesses a function F (as defined in section 3.1) of two formulae, and if he possesses one of these formula, then he is capable of possessing the other formula as well.

P6 $$\frac{P \ni K,\ P \ni X}{P \ni \{X\}_K,\ P \ni \{X\}_K^{-1}}$$

If a principal possesses a formula and a key then he is capable of possessing both the encryption and the decryption of the formula with the key.

P7 $$\frac{P \ni +K,\ P \ni X}{P \ni \{X\}_{+K}}$$

If a principal possesses a formula and a public key then he is capable of possessing the encryption of that formula with the key.

P8 $$\frac{P \ni -K,\ P \ni X}{P \ni \{X\}_{-K}}$$

If a principal possesses a formula and a private key then he is capable of possessing the decryption of that formula with the key.

A.3 Freshness Rules

For convenience, we use $P \mid\equiv \sharp(X,Y)$ to denote $P \mid\equiv \sharp(X)$ or $P \mid\equiv \sharp(Y)$.

F1 $$\frac{P \mid\equiv \sharp(X)}{P \mid\equiv \sharp(X,Y),\ P \mid\equiv \sharp(F(X))}$$

If P believes a formula X is fresh, then he is entitled to believe that any formula of which X is a component is fresh, and that a computationally feasible one-to-one function F of X is fresh.

F2 $$\frac{P \mid\equiv \sharp(X),\ P \ni K}{P \mid\equiv \sharp(\{X\}_K),\ P \mid\equiv \sharp(\{X\}_K^{-1})}$$

If P believes a formula X is fresh and possesses a key, then P is entitled to believe that the encryption and the decryption of X with the key are fresh.

F3 $$\frac{P \mid\equiv \sharp(X),\ P \ni +K}{P \mid\equiv \sharp(\{X\}_{+K})}$$

If P believes a formula X is fresh and possesses a public key, then P is entitled to believe that the encryption of X with that key is fresh as well.

F4 $$\frac{P \mid\equiv \sharp(X),\ P \ni -K}{P \mid\equiv \sharp(\{X\}_{-K})}$$

If P believes a formula X is fresh and possesses a private key, then P is entitled to believe that the decryption of X with that key is fresh as well.

F5 $$\frac{P \mid\equiv \sharp(+K)}{P \mid\equiv \sharp(-K)}$$

If P believes that a public key is fresh, then he is entitled to believe that the corresponding private key is fresh as well.

F6 $$\frac{P \mid\equiv \sharp(-K)}{P \mid\equiv \sharp(+K)}$$

If P believes that a private key is fresh, then he is entitled to believe that the corresponding public key is fresh as well.

F7 $$\frac{P \mid\equiv \phi(X),\ P \mid\equiv \sharp(K),\ P \ni K}{P \mid\equiv \sharp(\{X\}_K),\ P \mid\equiv \sharp(\{X\}_K^{-1})}$$

If P believes that a formula X is recognizable and P possesses a key K and believes it is fresh, then P is entitled to believe that the encryption and the decryption of X with K are fresh.

F8 $$\frac{P \mid\equiv \phi(X),\ P \mid\equiv \sharp(+K),\ P \ni +K}{P \mid\equiv \sharp(\{X\}_{+K})}$$

If P believes that a formula X is recognizable, and P possesses a public key and believes it is fresh then P is entitled to believe that the encryption of X with that public key is fresh.

F9 $$\frac{P \mid\equiv \phi(X),\ P \mid\equiv \sharp(-K),\ P \ni -K}{P \mid\equiv \sharp(\{X\}_{-K})}$$

If P believes that a formula X is recognizable and P possesses a private key and believes it is fresh, then P is entitled to believe that the decryption of X with that private key is fresh.

F10 $$\frac{P \mid\equiv \sharp(X),\ P \ni X}{P \mid\equiv \sharp(H(X))}$$

If P believes a formula X is fresh and he also possesses X, then he is entitled to believe that a one-way function of X is fresh.

F11 $$\frac{P \mid\equiv \sharp(H(X)),\ P \ni H(X)}{P \mid\equiv \sharp(X)}$$

If P possesses $H(X)$ and believes it to be fresh, then he is entitled to believe that X is fresh.

A.4 Recognizability Rules

R1 $$\frac{P \mid\equiv \phi(X)}{P \mid\equiv \phi(X,Y),\ P \mid\equiv \phi(F(X))}$$

If P believes a formula X is recognizable, then he is entitled to believe that any formula of which X is a component is recognizable, and that a computationally feasible function F of X is recognizable.

R2 $$\frac{P \mid\equiv \phi(X),\ P \ni K}{P \mid\equiv \phi(\{X\}_K),\ P \mid\equiv \phi(\{X\}_K^{-1})}$$

If P believes a formula X is recognizable and P possesses a key K, then P is entitled to believe that the encryption and the decryption of X with K are recognizable.

R3 $$\frac{P \mid\equiv \phi(X),\ P \ni +K}{P \mid\equiv \phi(\{X\}_{+K})}$$

If P believes a formula X is recognizable and P possesses a public key , then P is entitled to believe that the encryption of X with that key is recognizable.

R4 $$\frac{P \mid\equiv \phi(X),\ P \ni -K}{P \mid\equiv \phi(\{X\}_{-K})}$$

If P believes a formula X is recognizable and P possesses a private key, then P is entitled to believe that the decryption of X with that key is recognizable.

R5 $$\frac{P \mid\equiv \phi(X),\ P \ni X}{P \mid\equiv \phi(H(X))}$$

If P believes a formula X is recognizable and he also possesses X, then he is entitled to believe that a one-way function of X is recognizable.

R6 $$\frac{P \ni H(X)}{P \mid\equiv \phi(X)}$$

If P possesses formula $H(X)$ then he is entitled to believe that X is recognizable.

A.5 Message Interpretation Rules

A secret S used for identification purposes is denoted as $< S >$. This way it can be distinguished when other secrets are transmitted as data in the same message.

I1

$$\frac{P \lhd *\{X\}_K,\ P \ni K,\ P \mid\equiv P \overset{K}{\leftrightarrow} Q,\ P \mid\equiv \phi(X),\ P \mid\equiv \sharp(X,K)}{P \mid\equiv Q \mid\sim X,\ P \mid\equiv Q \mid\sim \{X\}_K,\ P \mid\equiv Q \ni K}$$

Suppose that for principal P all of the following conditions hold: (1) P receives a formula consisting of a X encrypted with key K and marked with a not-originated-here mark; (2) P possesses K; (3) P believes K is a suitable secret for himself and Q; (4) P believes formula X is recognizable; (5) P believes that K is fresh or that X is fresh.

Then P is entitled to believe that (1) Q once conveyed X; (2) Q once conveyed the formula X encrypted with K; (3) Q possesses K.

I2

$$\frac{P \lhd *\{X, < S >\}_{+K},\ P \ni (-K,S),\ P \mid\equiv \overset{+K}{\mapsto} P,\ P \mid\equiv P \overset{S}{\leftrightarrow} Q,\ P \mid\equiv \phi(X,S),\ P \mid\equiv \sharp(X,S,+K)}{P \mid\equiv Q \mid\sim (X, < S >),\ P \mid\equiv Q \mid\sim \{X, < S >\}_{+K},\ P \mid\equiv Q \ni +K}$$

Suppose that for principal P all of the following conditions hold: (1) P receives a formula consisting of X concatenated with S, encrypted with a public key and marked with a not-originated-here mark; (2) P possesses S and the corresponding private key; (3) P believes the public key is his own; (4) P believes S is a suitable secret for himself and Q; (5) P believes that X concatenated with S is recognizable; (6) P believes that at least one of $S, X, or + K$ is fresh.

Then P is entitled to believe that (1) Q once conveyed the formula X concatenated with S; (2) Q once conveyed the formula X concatenated with S and encrypted with the public key; (3) Q possesses the public key.

I3

$$\frac{P \lhd *H(X, < S >),\ P \ni (X,S),\ P \mid\equiv P \overset{S}{\leftrightarrow} Q,\ P \mid\equiv \sharp(X,S)}{P \mid\equiv Q \mid\sim (X, < S >),\ P \mid\equiv Q \mid\sim H(X, < S >)}$$

Suppose that for principal P all of the following conditions hold: (1) P receives a formula consisting of a one-way function of X and S marked with a not-originated-here mark; (2) P possesses S and X; (3) P believes S is a suitable secret for himself and Q; (4) P believes that either S or X is fresh.

Then P is entitled to believe that (1) Q once conveyed the formula X concatenated with S; (2) Q once conveyed the one-way function of X concatenated with S.

To understand I3 on the use of one-way functions, it helps to compare it with the previous two. Also, recall that according to our definition $Q \mid\sim (X,S)$ does not mean that Q necessarily included and transmitted X and S explicitly in any message, but rather that he conveyed X and S.

Postulates I4 and I5 hold only for RSA or other public-key schemes with a similar property to the one described in section 3.

I4

$$\frac{P \lhd \{X\}_{-K},\ P \ni +K,\ P \mid\equiv \overset{+K}{\mapsto} Q,\ P \mid\equiv \phi(X)}{P \mid\equiv Q \mid\sim X,\ P \mid\equiv Q \mid\sim \{X\}_{-K}}$$

Suppose for principal P, all of the following conditions hold: (1) P receives a formula consisting of X encrypted with a private key; (2) P possesses the corresponding public key; (3) P believes that public key is Q's; (4) P believes X is recognizable.

Then P is entitled to believe that (1) Q once conveyed the formula X; (2) Q once conveyed the formula consisting of X encrypted with the private key.

I5

$$\frac{P \lhd \{X\}_{-K},\ P \ni +K,\ P \mid\equiv \overset{+K}{\mapsto} Q,\ P \mid\equiv \phi(X),\\ P \mid\equiv \sharp(X, +K)}{P \mid\equiv Q \ni (-K, X)}$$

Suppose for principal P, all of the following conditions hold: (1) P receives a formula consisting of X encrypted with a private key; (2) P possesses the corresponding public key; (3) P believes that public key is Q's; (4) P believes X is recognizable; (5) P believes that either X or the public key is fresh.

Then P is entitled to believe that Q possesses the string consisting of the concatenation of X and the private key.

I6 $$\frac{P \mid\equiv Q \mid\sim X,\ P \mid\equiv \sharp(X)}{P \mid\equiv Q \ni X}$$

If P believes that Q once conveyed formula X and P believes that X is fresh, then P is entitled to believe that Q possesses X.

I7 $$\frac{P \mid\equiv Q \mid\sim (X, Y)}{P \mid\equiv Q \mid\sim X}$$

If P believes that Q once conveyed the formula consisting of the concatenation of X and Y, then P is entitled to believe that Q once conveyed X.

A.6 Jurisdiction Rules

J1 $$\frac{P \mid\equiv Q \mid\Longrightarrow C,\ P \mid\equiv Q \mid\equiv C}{P \mid\equiv C}$$

If P believes that Q is an authority on some statement C and that Q believes in C, then P ought to believe in C as well.

J2 $$\frac{P \mid\equiv Q \mid\Longrightarrow Q \mid\equiv *,\ P \mid\equiv Q \mid\sim (X \leadsto C),\\ P \mid\equiv \sharp(X)}{P \mid\equiv Q \mid\equiv C}$$

If P believes that Q is honest and competent, and P receives a message $X \leadsto C$ which he believes Q conveyed, then P ought to believe that Q really believes C.

J3 $$\frac{P \mid\equiv Q \mid\Longrightarrow Q \mid\equiv *,\ P \mid\equiv Q \mid\equiv Q \mid\equiv C}{P \mid\equiv Q \mid\equiv C}$$

If P believes that Q is honest and competent, and P believes that Q believes that Q believes in C, then P ought to believe that Q believes in C.

B "Never-Originated-Here" Messages

The "not-originated-here" notion in section 3.3 refers to messages in the current session. Some principals may be able to identify that certain messages were not originated by themselves in current or previous sessions. Suppose we use $P \mid\equiv \otimes(P)$ to denote that P believes he can identify such messages, then we can add the following postulates.

I1′ $$\frac{P \lhd \{X\}_K,\ P \ni K,\ P \mid\equiv P \overset{K}{\leftrightarrow} Q,\ P \mid\equiv \phi(X),\\ P \mid\equiv \otimes(P)}{P \mid\equiv Q \mid\sim X,\ P \mid\equiv Q \mid\sim \{X\}_K}$$

Suppose that for principal P all of the following conditions hold: (1) P receives a formula consisting of X encrypted with key K and marked with a not-originated-here mark; (2) P possesses K; (3) P believes K is a suitable secret for himself and Q; (4) P believes formula X is recognizable; (5) P believes that he can identify that a message was not originated by himself.

Then P is entitled to believe that Q once conveyed X and Q once conveyed X encrypted with K.

I2′

$$\frac{P \lhd \{X, <S>\}_{+K}, P \ni (S, -K),\ P \mid\equiv \overset{+K}{\mapsto} P,\\ P \mid\equiv P \overset{S}{\leftrightarrow} Q,\ P \mid\equiv \phi(X, S), P \mid\equiv \otimes(P)}{P \mid\equiv Q \mid\sim (X, <S>),\ P \mid\equiv Q \mid\sim \{X, <S>\}_{+K}}$$

Suppose that for principal P all of the following conditions hold: (1) P receives a formula consisting of X concatenated with S, encrypted with a public key and marked with a not-originated-here mark; (2) P possesses S and the corresponding private key; (3) P believes the public key is his own; (4) P believes S is a suitable secret for himself and Q; (5) P believes that X concatenated with X is recognizable; (6) P believes he can identify that a message was not originated by himself.

Then P is entitled to believe that Q once conveyed the formula X concatenated with S, and Q once conveyed the formula X concatenated with S encrypted with the public key.

I3′

$$\frac{P \lhd H(X, <S>),\ P \ni (X, S),\ P \mid\equiv P \overset{S}{\leftrightarrow} Q,\\ P \mid\equiv \phi(X, S),\ P \mid\equiv \otimes(P)}{P \mid\equiv Q \mid\sim (X, <S>),\ P \mid\equiv Q \mid\sim H(X, <S>)}$$

Suppose that for principal P all of the following conditions hold: (1) P receives a formula consisting of a one-way function of X and S marked with a not-originated-here mark; (2) P possesses S and X; (3) P believes S is a suitable secret for himself and Q; (4) P believes that S concatenated with X is recognizable; (5) P believes he can identify that a message was not originated by himself.

Then P is entitled to believe that Q once conveyed the formula X concatenated with S, and Q once conveyed the one-way function of X concatenated with S.

It is possible, and perhaps worthwhile, to refine the notion by associating such an assumption with particular formulae. Thus for example in I1$'$ above we would replace $P \mid\equiv \otimes(P)$ with $P \mid\equiv \otimes(\{X\}_K)$.

C <u>Semantics</u>

The semantics are similar to those in BAN [1]. The semantics are "operational". Principals develop new beliefs and accumulate possessions by applying inference rules to their present beliefs, possessions, and received messages.

- The local state of a principal P consists of two sets. A set of formulae $\mathcal{P}_P$ and a set of statements $\mathcal{B}_P$. Intuitively, $\mathcal{P}_P$ is the set of formulae the principal possesses and $\mathcal{B}_P$ is the set of beliefs of the principal. The sets have some closure properties, as reflected in the inference rules.

- A global state is a tuple containing the local states of all principals. Suppose s is a global state then s_P is the local state of P in s and $\mathcal{P}_P(s)$ and $\mathcal{B}_P(s)$ are the corresponding sets of possession and beliefs. The satisfaction relation between global states and statements is: $P \mid\equiv$ C holds in a state s if C $\in \mathcal{B}_P(s)$ and $P \ni X$ holds if $X \in \mathcal{P}_P(s)$. A set of statements holds in a given state if each of its members holds.

- A run is a finite sequence of states $s_0, \ldots, s_n$ where $\mathcal{B}_P(s_i) \subseteq \mathcal{B}_P(s_{i+1})$, and $\mathcal{P}_P(s_i) \subseteq \mathcal{P}_P(s_{i+1})$ for all $i \leq (n-1)$. That is, the belief set and the possess set cannot decrease during a run.

- A protocol is a finite sequence of n expressions of the form:

  $(P_1 \rightarrow Q_1 : X_1 \rightsquigarrow C_1), \ldots, (P_n \rightarrow Q_n : X_n \rightsquigarrow C_n)$

- In a run of a protocol all messages of the protocol are communicated. It is a run of length $n+1$ where

  $X_i \in \mathcal{P}_{Qi}(s_i) \cap \mathcal{P}_{Pi}(s_{i-1})$ and $C_n \in \mathcal{B}_{Pi}(s_{i-1})$, for all $i \leq n$.

- An annotation for the protocol holds in a run of the protocol if all of the statements in the annotation hold in the corresponding states. An annotation is valid if it holds in all runs of the protocol where the first set of the annotation – the assumptions – holds.

The Logic of Authentication Protocols ⋆

Paul Syverson[1] and Iliano Cervesato[2]

[1] Center for High Assurance Computer Systems
Naval Research Laboratory
Washington, DC 20375 — USA
`syverson@itd.nrl.navy.mil`

[2] Advanced Engineering and Sciences Division
ITT Industries, Inc.
2560 Huntington Avenue, Alexandria, VA 22303 — USA
`iliano@itd.nrl.navy.mil`

1 Introduction

The rationale of authentication has been a topic of study for about a decade and a half. First attempts at formal analysis of authentication protocols were not using logics per se, but were certainly logical. Millen's *Interrogator* [Mil84,MCF87] was a Prolog based tool specifically designed for authentication protocol analysis that functioned essentially as a special purpose model checker. Kemmerer used the general purpose formal specification language *Ina Jo* and an accompanying symbolic execution tool *Inatest* to specify and analyze protocols [Kem87].

We will focus on logics of authentication, beginning with BAN [BAN89a]. However, we will not only be discussing logics per se. We will also be looking at the 'rhyme and reason' of authentication, the attempts to formalize and define notions of authentication and to apply these. Thus, we will also be considering the logic of authentication in a broader sense.

We will not discuss (except incidentally) other formal methods that have been applied to authentication. In particular, we will not be describing process algebras, automata, automated tools such as theorem provers or model checkers. Some of these other approaches are discussed elsewhere in this volume. The remainder of this section will provide background on authentication protocols and introduce a running example.

1.1 Background on Authentication Protocols

In this section we present basic background on the concepts of authentication and its building blocks in cryptography. If every device communicating on behalf

⋆ This paper is based on a course Syverson taught at the 1st International School on Foundations of Security Analysis and Design (FOSAD'00) in Bertinoro, Italy in September 2000. Cervesato was a student there. The work of the first author was supported by ONR. The work of the second author was supported by NSF grant INT98-15731 "Logical Methods for Formal Verification of Software" and by NRL under contract N00173-00-C-2086.

of a person or other entity shared a secret key with every other such device, and these keys were never compromised, canceled, unsubscribed, or otherwise expired, then basic authentication protocols *might* be unnecessary. Clearly this is not even remotely the case. It has thus long been recognized that there must be some mechanism by which principals that do not share such a secret key, and may not even have any knowledge of each other beyond possibly an identifier, can establish a key for a secure communication session.

An *authentication protocol* is an exchange of messages having a specific form for authentication of principals using cryptographic algorithms. They typically have additional goals such as the distribution of session keys. *Symmetric-key cryptography* (also called *secret-key cryptography*) relies on the same key for both encryption and decryption. Classic examples include the data encryption standard, DES, and its recent successor, AES, the advanced encryption standard. (More details about cryptography can be found in any number of books [MvOV97,Sch96,Sti95].) *Public-key cryptography* is encryption and decryption using different keys (also called asymmetric cryptography). The most well known example is RSA. A *public key* is so-called because it is generally available to anyone. Corresponding to the public key is a *private key*, stereotypically known only to one principal. The private key is used to decrypt the message. Because it is uniquely bound to an individual a private key can also be used for a *digital signature* on a message. Typically, different keys and different algorithms are used for decryption and digital signatures. For digital signatures, the public key is used to verify that the signature is that of the principal bound to the public key. Binding is usually accomplished by means of a certificate, typically a message asserting such binding, containing an indicator of timeliness and signed by a well-known trusted principal (server).

Security protocols may have any number of intended purposes. Some exotic examples are voting, fair exchange of goods or contracts, non-repudiation, and anonymous communication. We will focus on authenticated establishment of session keys, which is typically necessary for the running of security protocols for most other purposes. Authentication is essentially assurance of who you are talking to. This can be made more specific in any number of ways: for example, you may want to make sure that those obtaining a session key are who they say they are, make sure that someone who has the key is currently on line, make sure that the principal you think has the key does have it, make sure that the principal with whom you think you share the key also thinks he is sharing it with you, etc. We will go into more detail on these points in Section 4. For now the basic intuition should suffice.

If a protocol is used for some security purpose, this implies an adversary against which the protocol is secure. The standard adversary for formal analysis of security protocols was introduced by Dolev and Yao in 1983 and is commonly known as the Dolev-Yao adversary [DY83]. It is a very strong adversary, much stronger than is typically assumed for secure distributed computation as in, e.g., Byzantine agreement. In the Dolev-Yao case, all messages sent from any honest principal to any other must pass through the adversary. The adversary can read, alter, and redirect any and all messages. However, encryption is treated as a black box. The adversary can only decrypt a message if she has the right keys. She can only compose new messages from keys and messages that she already possesses. In particular, she cannot perform any statistical or other cryptanalytic attacks. Other common terms for the adversary include: attacker, penetrator, spy, intruder, enemy, and eavesdropper. Eavesdroppers are typically considered as only passive. But, any adversary is often referred to as 'Eve'.

1.2 Running Example

In this section, we introduce an example of an authentication protocol that will also be discussed in later sections. Eve's honest counterparts are traditionally named 'Alice' and 'Bob'. The other main principal in this protocol is the server. Alice and Bob are assumed to share keys (typically called 'long-term keys') with the server. Besides the obvious symbols for Alice (A), Bob (B), the server (S), and the keys they share (k_{AS}, k_{AS}, k_{AB}, etc.), the protocol introduces us to nonces, i.e., random unpredictable values generated by a principal and included in messages so that she can tell any messages later received and containing her nonce must have been produced after she generated and sent the nonce. A nonce generated by Alice is written 'n_A'. The session key that the server generates for Alice and Bob is k_{AB}. Encryption of a message, M, using key k is written '$\{M\}_k$'.

Protocol 1 (Needham-Schroeder Shared-Key) [NS78]

Message 1	$A \to S:$	A, B, n_A
Message 2	$S \to A:$	$\{n_A, B, k_{AB}, \{k_{AB}, A\}_{k_{BS}}\}_{k_{AS}}$
Message 3	$A \to B:$	$\{k_{AB}, A\}_{k_{BS}}$
Message 4	$B \to A:$	$\{n_B\}_{k_{AB}}$
Message 5	$A \to B:$	$\{n_B - 1\}_{k_{AB}}$

In this protocol, Alice indicates to the server that she would like to talk to Bob and includes a nonce, n_A. The server S sends her a message encrypted with the key they share. This message contains her nonce n_A (so that she knows the message is fresh), Bob's identifier (so she knows that this is indeed for a session between her and Bob), the session key k_{AB}, and an encrypted submessage $\{k_{AB}, A\}_{k_{BS}}$ to be forwarded to Bob. Alice decrypts this message and forwards the submessage to Bob. He decrypts it and sends Alice a nonce n_B encrypted with the session key to show that he has the session key and to check that she does. She decrypts the message, subtracts one from the nonce, re-encrypts, and sends it back to Bob, completing the protocol. In the next section, we will set out a logic for analyzing protocols such as this.

1.3 Organization

In Section 2 we will describe BAN logic, its rules and some limitations and revisions. In Section 3 we will describe one of BAN's successors, SVO. Both of those sections will include an analysis of the running example. We will then proceed in Section 4 to the 'rhyme and reason', presenting formal and informal goals of authentication and attempts to define it. We will turn to some of the other aspects of authentication protocols in Section 5, where we will look at design principles and properties that arise from them, such as fail-stop properties. We will also look at ways in which these have been combined with the logical approach described in the earlier sections. Another approach to connecting goals and logics is considered in Section 6, in which a logical language is used to express requirements that are evaluated on a semantic level rather than with a logic for that language. Semantics are further tied to the earlier sections by assigning meanings to BAN constructs in the strand space model of authentication protocols [THG97,THG98b]. Finally, in Section 7 we say a few words about the future of formal analysis of authentication protocols.

2 BAN Logic

In this section we present an overview of BAN logic.[1] Specifically, we introduce the concepts, notation, and rules of BAN, after which, we will give some sample analyses. We will end the section by describing some of the extensions to BAN.

BAN is a logic of belief. The intended use of BAN is to analyze authentication protocols by deriving the beliefs that honest principals correctly executing a protocol can come to as a result of the protocol execution. For example, Alice might come to believe that a key she has received from a server is a good key for a communication session with Bob. What 'good' means here will be discussed below. The approach is to "idealize" the messages in the protocol specification into logical formulae. For example, if a server sends Alice a session key inside an encrypted message, the key might be replace by a formula that means that the key is good. We could then draw inferences based on Alice's ability to decrypt the key and other assumptions that might ultimately lead to the conclusion that she believes that the received key is good for talking with Bob.

BAN has been highly successful in uncovering protocol flaws, needed assumptions, etc., and it is relatively easy to use. A clear motivation in BAN is the math-

[1] 'BAN' is derived from the names of its authors, Burrows, Abadi and Needham. It is the first in a family of eponymous authentication logics. Versions of this logic occurred in many places. The first presentation in a public forum was at TARK in March of 1988 [BAN88]. It was also presented at the first CSFW in June of 1988. A revised and expanded version of the logic was given at SOSP in December of 1989 [BAN89b]. Journal versions of this appeared in *ACM TOCS* [BAN90a] and in *Proceedings of the Royal Society of London* [BAN89c]. The *TOCS* paper is an abbreviated version of the same material. The *Proc. Royal Society* paper is the one typically cited by the authors. Our primary source is the the Digital Systems Research Center report [BAN89a], and all descriptions of BAN are drawn from it. This SRC report originated in February 1989 and was revised in February 1990 to include "The Scope of a Logic of Authentication", which first appeared at a DIMACS workshop in October 1989 [BAN90c]. The February 1989 version of the SRC report comprises the *Proc. Royal Society* Paper.

ematician's credo to make only the needed distinctions and no more. Thus for example, the authors simplify reasoning about time by only distinguishing past and present epochs. These are not determined by the ephemeral current instant but are constant, set once and for all. This gives rise to one of BAN's central concepts, freshness. The other central concept is the aforementioned goodness of keys.

2.1 BAN Notation

We note at this point that the notation we will use is not that of [BAN89a]. Rather it is largely the notation introduced in [AT91] (with the public key notation introduced in [vO93]). It is closer to plain English than the original BAN notation, hence a bit more intuitive. For example, compare the following expressions (the first is original BAN notation):

$$P \models Q \mid\!\sim \#(X) \quad \text{vs.} \quad P \textit{ believes } Q \textit{ said fresh}(X)$$

The language of BAN consists of the following expressions:

P believes X : P may act as if X is true.

P received X : P has received a message containing X, and P can get X from the message; this may require decryption.

P said X : P has sent a message containing X at some prior point, and P believed X and understood that he was sending X at that time.

P controls X : P has jurisdiction on X, i.e., it should be trusted concerning X.

fresh(X) : (Read 'X is fresh'.) X has not been sent in any message prior to the current protocol run.

$\boldsymbol{P \xleftrightarrow{k} Q}$: (Read 'k is a good key for P and Q'.) k will never be discovered by any principal but P, Q, or a principal trusted by P or Q. (The last case is necessary, since the server often sees, indeed generates, k.)

$\boldsymbol{PK(P,k)}$: (Read 'k is a public key of P'.) The secret key, k^{-1}, corresponding to k will never be discovered by any principal but P or a principal trusted by P.

$\boldsymbol{\{X\}_k}$: Short for "$\{X\}_k$ *from* P" (Read 'X encrypted with k (from P)'.) This is the notation for encryption. Principals can recognize their own messages. Encrypted messages are uniquely readable and verifiable as such by holders of the right keys.

In all of these expressions, "X" is either a message or a formula. As we will see, every formula can be a message, but not every message is a formula.

2.2 BAN Rules

In an analysis, the protocol is first idealized into messages containing assertions, then assumptions are stated, and finally conclusions are inferred based on the assertions in the idealized messages and those assumptions. The rules to do so are now given.

The first rule is called "message meaning". It and "nonce verification" are the central rules of BAN.

Message Meaning

$$\frac{P \textit{ believes } P \xleftrightarrow{k} Q \qquad P \textit{ received } \{X\}_k}{P \textit{ believes } Q \textit{ said } X}$$

"If P receives X encrypted with k and if P believes k is a good key for talking with Q, then P believes Q once said X." [BAN89a]

Note that this rule does not tell us anything about what submessage(s) P can extract from an encrypted message. That will come below under **Receiving Rules**. Rather, this rule tells us what P can discern about who sent the message. In applying symmetric keys, there is no explicit distinction between signing and encryption. (In BAN, there is also no distinction when applying public keys. Both signing and encryption are represented by $\{X\}_k$. The distinction is implicit in the notation for the key used: k or k^{-1}.)

There is also a public-key version of message meaning. The implied "*from*" field in the shared-key case would be redundant in the public-key case since it is implicit in the meaning of the notation for binding a public key to a principal who originated the signed message.

$$\frac{P \textit{ believes } PK(Q,k) \qquad P \textit{ received } \{X\}_{k^{-1}}}{P \textit{ believes } Q \textit{ said } X}$$

Nonce Verification

$$\frac{P \textit{ believes fresh}(X) \qquad P \textit{ believes } Q \textit{ said } X}{P \textit{ believes } Q \textit{ believes } X}$$

This rule allows promotion from the past to the present (something said some time in the past to a present belief). In order to be applied, X should not contain any encrypted text. This rule is the only way for such promotion to occur. Since principals are all assumed to be honest and competent with respect to following the protocol, it makes sense that anything that a principal said recently should be something that he believes. That is, it makes sense for assertions, but what if X is a nonce, n? Obviously, it is a stretch of the intuitive meaning of belief to say that Bob believes a nonce. It is not necessary that this technical use respect all of our intuitions. The goal of the logic is to provide something useful for the analysis of authentication protocols, not to formalize reasoning about ordinary belief. Nonetheless, [BAN89a] suggests introducing a "has recently said" operator. This was in fact done by many later authors;

although the motivation may have had more to do with making the belief part of the logic conform more closely to traditional modal logics of knowledge and belief.

Jurisdiction

$$\frac{P \textit{ believes } Q \textit{ controls } X \qquad P \textit{ believes } Q \textit{ believes } X}{P \textit{ believes } X}$$

The jurisdiction rule is what allows inferences that a principal believes a key is good, even though it is a random string that he has never seen before. An important thing to keep in mind about jurisdiction is the strength of *controls* statements. If P *controls* X, then P cannot make a mistake in asserting X. In BAN, this is somewhat tempered by only allowing inferences , e.g., from Alice's belief that the server controls $A \xleftrightarrow{k_{AB}} B$ to Alice's belief that $A \xleftrightarrow{k_{AB}} B$. In other words, inferences cannot be made about whether a key is actually good, but only about whether a key is believed to be good. Later logics, in particular AT and SVO, by separating off the axioms of belief, lose this tempering in principle. However, in practice this makes little difference.

Note that quantifiers are implicit. So "A *believes* S *controls* $A \xleftrightarrow{k} B$" is implicit for "A *believes* $\forall k.(S$ *controls* $A \xleftrightarrow{k} B)$". Of course leaving things implicit leads in principle to some ambiguities. For example, "A *believes* $\forall k.(S$ *controls* B *controls* $A \xleftrightarrow{k} B)$" vs. "A *believes* S *controls* $\forall k.(B$ *controls* $A \xleftrightarrow{k} B)$".

In practice, these questions do not usually arise in basic authentication protocols because nested assertions of jurisdiction are rarely found. As in other things, the BAN approach is to ignore complications not encountered in practice. If it should become necessary to be explicit, however, then they do so. For example, in [ABKL90], nested *controls* statements occur and the quantifiers in those statements are made explicit.

Belief Conjuncatenation

$$\frac{P \textit{ believes } X \qquad P \textit{ believes } Y}{P \textit{ believes } (X, Y)}$$

$$\frac{P \textit{ believes } Q \textit{ believes } (X, Y)}{P \textit{ believes } Q \textit{ believes } X} \qquad \frac{P \textit{ believes } Q \textit{ said } (X, Y)}{P \textit{ believes } Q \textit{ said } X}$$

The obvious rules apply to beliefs concerning concatenations of messages/conjunctions of formulae. We have chosen the neologistic mouthful 'conjuncatenation' to again reinforce the point that BAN makes only the distinctions it needs. In this case, concatenations of messages are not distinguished from conjunctions of formulae: both are represented as (X, Y) in the above rules. Also, following the lead of [BAN89a], we do not list all of the rules; we give only a representative sampling. For example, we will not state versions of the last two rules where the conclusions are replaced by P *believes* Q *believes* Y and P *believes* Q *said* Y.

Freshness Conjuncatenation

$$\frac{P \textit{ believes } \textit{fresh}(X)}{P \textit{ believes } \textit{fresh}(X, Y)}$$

For some inexplicable reason, this is a commonly misunderstood BAN rule. Some try to deny it; others try to assert the converse rule. Be wary of these mistakes. If X is fresh, then any message containing X is fresh in virtue of having X in it. But, (X, Y) being fresh tell us nothing about the freshness either of X by itself or of Y by itself (because the whole may be fresh in virtue of the other part).

Receiving Rules: Seeing is Receiving

$$\frac{P \textit{ believes } P \overset{k}{\longleftrightarrow} Q \qquad P \textit{ received } \{X\}_k}{P \textit{ received } X} \qquad\qquad \frac{P \textit{ received } (X, Y)}{P \textit{ received } X}$$

A principal receiving a message also receives submessages he can uncover. Here is another clever BAN fusion, one that is lost a little in our more English-like notation: in BAN the symbol for receiving, '$\triangleleft$', is used to reason about what is visible. Thus, what a principal possesses is not distinguished from what he has received in some message. Virtually all successors to BAN distinguished the two; yet BAN is able to analyze a large number of protocols without this distinction.

This completes our listing of the rules of BAN. We now describe how to use BAN to analyze a protocol.

2.3 BAN Protocol Analysis

There are four steps to a protocol analysis using BAN.

1. Idealize the protocol.
2. Write assumptions about the initial state.
3. Annotate the protocol: For each message transmission "$P \to Q : \ M$" in the protocol, assert Q *received* M.
4. Use the logic to derive the beliefs held by protocol principals.

As an example, we will go through a BAN analysis of the Needham-Schroeder Shared-Key protocol of Section 1.2.

Example: Analysis of Needham-Schroeder Shared-Key (NSSK)

The first step is to put the protocol into idealized form.

Idealized Needham-Schroeder Shared-Key [BAN89a]

Message 2 $S \to A : \ \{n_A, A \overset{k_{AB}}{\longleftrightarrow} B, \textit{fresh}(k_{AB}), \{A \overset{k_{AB}}{\longleftrightarrow} B\}_{k_{BS}}\}_{k_{AS}} \textit{ from } S$

Message 3 $A \to B : \{A \overset{k_{AB}}{\longleftrightarrow} B\}_{k_{BS}} \textit{ from } S$

Message 4 $B \to A : \{n_B, A \overset{k_{AB}}{\longleftrightarrow} B\}_{k_{AB}} \textit{ from } B$

Message 5 $A \to B : \{n_B, A \overset{k_{AB}}{\longleftrightarrow} B\}_{k_{AB}} \textit{ from } A$

Note that the first message of the protocol is omitted in the idealized form. In a BAN idealization, plaintext from the protocol is omitted. Next note the *from* fields. It is always assumed that principals can recognize their own messages. Thus, with a shared key, if a recipient can decrypt a message, she can tell who it is from. As this is often implicitly clear, the *from* field is often omitted from the protocol idealization. What is inside the encrypted messages is also altered. Specifically, the key k_{AB} is replaced by assertions about it. So, Message 2, idealized, is an encrypted message for Alice from the Server that contains a nonce, an assertion that k_{AB} is fresh, an assertion that k_{AB} is good for talking to Bob, and an encrypted message to be forwarded to Bob. Note also that in the last message $n_B - 1$ is changed to just n_B. This is because the purpose of subtracting 1 from the nonce is to differentiate it from Message 4. The differentiation is reflected in the idealization in the *from* field. It would reduce informal interpretation to simply leave $n_B - 1$ in the idealized protocol. But, BAN has no direct rule to infer the freshness of $n_B - 1$ from the freshness of n_B. So, the change is necessary. This was changed in SVO, as we shall see below.

Once the idealization has been made, assumptions are stated.

NSSK Initial State Assumptions

P1. A *believes* $A \xleftrightarrow{k_{AS}} S$
P2. B *believes* $B \xleftrightarrow{k_{BS}} S$

P3. A *believes* S *controls* $A \xleftrightarrow{k} B$
P4. B *believes* S *controls* $A \xleftrightarrow{k} B$
P5. A *believes* S *controls fresh*$(A \xleftrightarrow{k} B)$

P6. A *believes fresh*(n_A)
P7. B *believes fresh*(n_B)

Most of the assumptions should be self-explanatory. P1 and P2 express the belief in the quality of the long term keys. (Notice that we make no corresponding assumptions about what S believes. It would be natural to do so, but we have omitted them because, in this case, they are not needed in the derivations that follow.) P3 through P5 give the assertions on which Alice and Bob believe that the server has jurisdiction. P6 and P7 tell us that each principal believes that his/her random value is fresh.

NSSK Annotated Protocol

The annotation states assumptions based on the messages in the idealized protocol. It can be read directly from the idealization.

P8. A *received* $\{n_A, A \xleftrightarrow{k_{AB}} B, \textit{fresh}(k_{AB}), \{A \xleftrightarrow{k_{AB}} B\}_{k_{BS}}\}_{k_{AS}}$
P9. B *received* $\{A \xleftrightarrow{k_{AB}} B\}_{k_{BS}}$ *from* S
P10. A *received* $\{n_B, A \xleftrightarrow{k_{AB}} B\}_{k_{AB}}$ *from* B
P11. B *received* $\{n_B - 1, A \xleftrightarrow{k_{AB}} B\}_{k_{AB}}$ *from* A

This completes the assumptions needed to analyze the protocol. In the derivations below, every line is followed by a justification, i.e., the rule by which it was derived and the premise(s) and/or derived formula(e) used in its derivation.

NSSK Derivations

1. A *believes* S *said* $(n_A, A \xleftrightarrow{k_{AB}} B, \mathit{fresh}(A \xleftrightarrow{k_{AB}} B), \{A \xleftrightarrow{k_{AB}} B\}_{k_{BS}})$
   By Message Meaning using P1, P8.
2. A *believes* $\mathit{fresh}(n_A, A \xleftrightarrow{k_{AB}} B, \mathit{fresh}(A \xleftrightarrow{k_{AB}} B), \{A \xleftrightarrow{k_{AB}} B\}_{k_{BS}})$
   By Freshness Conjuncatenation using 1, P6.
3. A *believes* S *believes* $(n_A, A \xleftrightarrow{k_{AB}} B, \mathit{fresh}(A \xleftrightarrow{k_{AB}} B), \{A \xleftrightarrow{k_{AB}} B\}_{k_{BS}})$
   By Nonce Verification using 2, 1.
4. A *believes* S *believes* $(A \xleftrightarrow{k_{AB}} B)$
   By Belief Conjuncatenation using 3.
5. A *believes* S *believes* $(\mathit{fresh} A \xleftrightarrow{k_{AB}} B)$
   By Belief Conjuncatenation using 3.
6. A *believes* $(A \xleftrightarrow{k_{AB}} B)$
   By Jurisdiction using 4, P3.
7. A *believes* $\mathit{fresh}(A \xleftrightarrow{k_{AB}} B)$
   By Jurisdiction using 4, P5.

We have derived Alice's belief in the goodness and in the freshness of k_{AB}. We now turn to Bob.

8. B *believes* S *said* $A \xleftrightarrow{k_{AB}} B$
   By Message Meaning using P2, P9.

With the assumptions we have made so far this is all we are able to derive with respect to Bob's belief in the goodness of k_{AB}. Unlike Alice, Bob has sent no nonce at this point in the protocol. The only way for us to move further is if we assume that Bob believes something he has received is fresh. We therefore add the assumption that Bob believes that the assertion $A \xleftrightarrow{k_{AB}} B$ is fresh.

P12. B *believes* $\mathit{fresh}(A \xleftrightarrow{k_{AB}} B)$

This is different than our earlier freshness assumptions since those were all based on values that the believing principal had herself generated. This one expresses Bob's belief that a random value someone else has generated is fresh. We will return to this odd assumption below after we complete the derivations.

9. B *believes* S *believes* $A \xleftrightarrow{k_{AB}} B$
   By Nonce Verification using P12, 8.
10. B *believes* $A \xleftrightarrow{k_{AB}} B$
   By Jurisdiction using P4, 9.

Unlike for A, for B we were forced to assume B *believes* $\mathit{fresh}(A \xleftrightarrow{k_{AB}} B)$ since we were unable to derive it. We have now derived Alice and Bob's first order beliefs in the goodness and freshness of k_{AB}. We next derive their second order beliefs.

11. *A believes B said* $(n_B, A \overset{k_{AB}}{\longleftrightarrow} B)$

    By Message Meaning using 6, P10.

12. *A believes fresh*$(n_B, A \overset{k_{AB}}{\longleftrightarrow} B)$

    By Freshness Conjuncatenation using 7.

13. *A believes B believes* $(n_B, A \overset{k_{AB}}{\longleftrightarrow} B)$

    By Nonce Verification using 12, 11.

14. *A believes B believes* $A \overset{k_{AB}}{\longleftrightarrow} B$

    By Belief Conjuncatenation using 13.

By similar reasoning, we can obtain *B believes A believes* $A \overset{k_{AB}}{\longleftrightarrow} B$ but with an important difference. Since Bob believes that $A \overset{k_{AB}}{\longleftrightarrow} B$ is fresh, there is no need for n_B in order for him to reach this conclusion. The only role n_B plays in the protocol is to differentiate Message 4 from Message 5. It does not need to be fresh for that. This makes the assumption that Bob believes $A \overset{k_{AB}}{\longleftrightarrow} B$ to be fresh stand out all the more. We now illustrate that the dubious nature of this assumption is not just an artifact of the analysis.

The Denning-Sacco Attack

In 1981, Denning and Sacco showed how the Needham-Schroeder Shared-Key protocol could be attacked if an attacker compromised an old session key [DS81]. In the attack specification 'E_A' is the attacker masquerading as A.

Message 3	$E_A \to B : \{k_{AB}, A\}_{k_{BS}}$
Message 4	$B \to E_A : \{n'_B\}_{k_{AB}}$
Message 5	$E_A \to B : \{n'_B - 1\}_{k_{AB}}$

The attack relies on the fact that Bob has no way to actually be assured that Message 3 is fresh. So, an attacker could spend whatever time is needed to break the session key k_{AB}. As long as she can do so within the lifetime of k_{BS}, then she can run the above attack. Bob will then think he has confirmed sharing k_{AB} with Alice, when in reality Alice is not present and the attacker knows the key. The attack is not directly uncovered by a BAN analysis of the protocol; rather the analysis shows that the protocol cannot achieve any sort of authentication for Bob without making the dubious assumption that underlies the attack.

This concludes our basic introduction to BAN logic and its use in protocol analysis. In the remainder of this section, we will set out some of the issues that led researchers to expand and modify BAN and the analysis technique.

2.4 The Nessett Protocol

In 1990, Nessett introduced the following simple example that "demonstrates that a significant flaw exists in the Burrows, Abadi and Needham logic" [Nes90].

Protocol 2 (Nessett) [Nes90]

Message 1 $A \to B: \quad \{n_A, k_{AB}\}_{k_A^{-1}}$

Message 2 $B \to A: \quad \{n_B\}_{k_{AB}}$

In the first message, Alice encrypts a session key using a private key, the public cognate of which is assumed to be widely known. Bob then send her a handshake value encrypted with the session key, k_{AB}. Of course the key is not at all good for communication between Alice and Bob because anyone can extract it from the first message and use it for communication. But, structurally the protocol is the same as one where the first message was encrypted with a good key. Here is Nessett's idealization of the protocol followed by the corresponding annotation, then by the initial state assumptions Nessett presents.

Idealized Nessett Protocol

Message 1 $A \to B: \quad \{n_A, A \overset{k_{AB}}{\longleftrightarrow} B\}_{k_A^{-1}}$

Message 2 $B \to A: \quad \{A \overset{k_{AB}}{\longleftrightarrow} B\}_{k_{AB}}$

Annotation Premises

P1. B *received* $\{n_A, A \overset{k_{AB}}{\longleftrightarrow} B\}_{k_A^{-1}}$

P2. A *received* $\{A \overset{k_{AB}}{\longleftrightarrow} B\}_{k_{AB}}$

Initial State Assumptions

P3. B *believes* $PK(k_A, A)$

P4. A *believes* $A \overset{k_{AB}}{\longleftrightarrow} B$

P5. A *believes fresh*$(A \overset{k_{AB}}{\longleftrightarrow} B)$

P6. B *believes fresh*(n_A)

P7. B *believes* A *controls* $(A \overset{k}{\longleftrightarrow} B)$

Note that Nessett assumes Bob to believe that n_A is fresh. Therefore, it is more naturally thought of as a timestamp than a nonce. We have used this notation to more closely follow Nessett[2]. Based on this idealization and set of assumptions, Nessett is makes the following derivations.

Nessett Protocol Derivations

1. B *believes* A *said* $(n_A, A \overset{k_{AB}}{\longleftrightarrow} B)$
   By Message Meaning using P3, P1.

[2] Specifically, he used "$\mathrm{N_A}$".

2. *B believes fresh*$(n_A, A \overset{k_{AB}}{\longleftrightarrow} B)$
   By Freshness Conjuncatenation using P6.
3. *B believes A believes* $(n_A, A \overset{k_{AB}}{\longleftrightarrow} B)$
   By Nonce Verification using 2, 1.
4. *B believes A believes* $A \overset{k_{AB}}{\longleftrightarrow} B$
   By Belief Conjuncatenation using 3.
5. *B believes* $A \overset{k_{AB}}{\longleftrightarrow} B$
   By Jurisdiction using P7, 4.

This completes the derivations for Bob. We now derive Alice's second order belief in the goodness of k_{AB}. (Her first order belief was assumed.)

6. *A believes B said* $A \overset{k_{AB}}{\longleftrightarrow} B$
   By Message Meaning using P4, P2.
7. *A believes B believes* $A \overset{k_{AB}}{\longleftrightarrow} B$
   By Nonce Verification using P5, 6.

Nessett's Critique of BAN

Using BAN, one can derive all of the typical BAN authentication goals for both Alice and Bob via the Nessett protocol—as we have just done. This shows that, according to BAN, k_{AB} is a good session key. But, k_{AB} is not a good key. Ergo, BAN is flawed. Nessett traces the source of the "flaw" to the scope of BAN. It addresses who gets and acknowledges a key (authentication), but it does not address who should not get a key (confidentiality).

Burrows et al. respond to Nessett in [BAN90b] by noting that their paper explicitly limits discussion to authentication of honest principals. They explicitly do no attempt to detect unauthorized release of secrets. Since Alice publishes k_{AB} in first message, Nessett's assumption *A believes* $A \overset{k_{AB}}{\longleftrightarrow} B$ is inconsistent with these stated restrictions. And, from absurd assumptions come absurd conclusions. The logic does not preclude ridiculous assumptions.

> "This seems fair enough; no logic protects against the assumption of bad premises. All one can reasonably ask is that, if the premises are true, then the conclusion is also true. On the other hand, Nessett could counter that illustrative counterexamples are supposed to be obvious; that's what makes them illustrative. If one wants to demonstrate that an argument form is invalid, one constructs an argument with that form that is clearly invalid. It is hardly fair to say that this shows nothing because it's an obviously invalid argument. That's the point. The danger is that one might use the form to justify an invalid argument that is not obviously so. The question remains: On what (extralogical) basis do we decide what goes into the premise set? For this case, Burrows et al. would no doubt contend that one includes a principal's belief in the goodness of a key only if she has reason to believe that it is good and no reason to think that it is not. Of course that is what we would like to do, but it is also what we are trying to determine how to do." [Syv91]

Certainly it is much too strong to say that the Nessett example shows the logic to be flawed. It does highlight a place where one is expected to rely purely on the intuitive reasonableness of assumptions. However, it has not shown that this results in either a logical error or a practical vulnerability.

Still, it would be nice to have a way to capture either formally or at least rigorously, the difference between Nessett-type protocols and those not flawed in this way. Alice's action is inconsistent with the meaning of $A \textit{ believes } A \overset{k_{AB}}{\longleftrightarrow} B$. What is needed is a way to reflect this mathematically [Syv91,Syv92]. Suppose we could derive $A \textit{ believes } C \textit{ has } k_{AB}$ (for arbitrary C). Increasing expressiveness would let us formally demonstrate this.

2.5 Expanding beyond BAN

In 1990, Gong, Needham, and Yahalom, introduced a new logic [GNY90] that extended BAN. This logic came to be known as *GNY*, following the precedent set by 'BAN'. In it one can represent possession of keys. So $A \textit{ believes } C \textit{ has } k_{AB}$ can be expressed, and possibly derived. GNY also distinguishes available messages from received messages. Other important contributions include formalizing a principal's distinction of his own generated messages from others. Analysis in GNY also leaves cleartext in idealized protocols, rather than assuming that it cannot play a role in authentication. While not specifically formulated as a response to Nessett, this logic allows the expression of key possession and thus can express formally that with which the dubious assumption is supposed to be inconsistent. By itself this does not guarantee that the needed key possession is derivable, nor does it directly express the inconsistency.

Another response to Nessett that comes closer to directly reflecting the inconsistency of meaning was first given by Abadi and Tuttle in [AT91]. Specifically, they presented a BAN-like logic that possessed an independently motivated account of meaning in the form of a model-theoretic semantics. This allows one to rigorously assess the truth of assumptions (consistent with a protocol). Specifically, the *AT* logic was closer to traditional modal logics than BAN, provided a detailed model of computation, had a soundness result with respect to the model, and was also more expressive (e.g., could express key possession). A traditional semantics was much of the motivation for this work. While the published soundness result had a mistaken assumption in it, this was a large step towards putting BAN on a footing both firmer and logically more traditional. We will return to semantics below.

Another important limitation on BAN is the type of protocols to which it can be applied. Diffie-Hellman protocols underly much of modern authenticated key distribution. For example, they underly the IETF standard Internet Key Exchange (IKE) protocol [DH99], as well as both SSL and TLS—what your Web browser uses when it makes a secure connection. Thus, being able to reason about such protocols would be quite useful. Paul van Oorschot's VO logic [vO93] was designed primarily to add this capability. It is an extension of GNY that can be used to reason about Diffie-Hellman type key agreement. In addition, [vO93] extended the lexicon of formally stated authentication properties, and formalized reasoning about confirmed possession of secrets. We will return to those points in Section 4. Right now we turn to a logic that attempts to comprise all of the advantages that VO and the other BAN logics introduce.

3 SVO Logic: Unifying the predecessors

In the last section, we saw some of the limitations of BAN, and the extensions and variants that were intended to overcome them. Each of the logics, GNY, AT, and VO brought a distinct addition. In response to this diversity, Syverson and van Oorschot devised a logic, SVO, that was intended to unify the above predecessors [SvO94,SvO96]. The intent was not simply to patch on new notation and rules adequately expressive to capture the additional scope of these logics. This would be both inelegant and potentially unsound. Rather, the intent was to produce a model of computation and a logic that was sound with respect to that model while still retaining the expressiveness of the various BAN extensions.

3.1 SVO Notation

SVO uses the notation already introduced for BAN, with the following main additions:

$\neg\varphi$: Negations of formulae are added to the language.

$\boldsymbol{P}$ ***says*** $\boldsymbol{X}$: X is a message P said recently. Like BAN's "P *said* X" but P must have said X since the beginning of current epoch.

$\boldsymbol{P}$ ***has*** $\boldsymbol{X}$: X is a message P can see. This includes messages

- Initially available to P,
- Received by P,
- Freshly generated by P, and
- Constructible by P from the above.

The original BAN idea of a public key might be expressed as '$PK(P, k)$', meaning that k is a public key of P — the matching secret key k^{-1} will never be discovered by any principal but P or a principal trusted by P. In SVO, this is refined to cover different types of public key functionality.

$\boldsymbol{PK_\psi(P, k)}$: k is a public ciphering key of P. Only P can read messages encrypted with k.

$\boldsymbol{PK_\sigma(P, k)}$: k is a public signature key of P. The key k verifies that messages signed by k^{-1} are from P.

$\boldsymbol{PK_\delta(P, k)}$: k is a public key-agreement key of P. A Diffie-Hellman key formed with k is shared with P.

$\boldsymbol{\lfloor X \rfloor_k}$: X signed with key k. Along with the differentiation of the various types of public keys, SVO distinguishes signature from encryption.

$\boldsymbol{\{X\}_k}$: This is no longer short for '$\{X\}_k$ *from* P'. In SVO, it is not assumed that principals can recognize their own messages. But it is still assumed that encrypted messages are uniquely readable and verifiable as such by holders of the right keys.

$\langle \boldsymbol{X} \rangle_{*\boldsymbol{P}}$: X according to P. [3] This is used for messages that P doesn't know or recognize (e.g., $\{X\}_k$ where P does not know k). P will nonetheless recognize $\langle \{X\}_k \rangle_{*P}$ as the same thing if received again, even if he cannot tie it back to any plaintext.

X from P : X was sent by P.

3.2 SVO Rules

Like AT, SVO is much more of a traditional axiomatic style logic than BAN, GNY or VO. As such there are only two rules.

Modus Ponens

$$\frac{\varphi \qquad \varphi \rightarrow \psi}{\psi}$$

Necessitation

$$\frac{\vdash \varphi}{\vdash P \textit{ believes } \varphi}$$

'$\vdash$' is a metalinguistic symbol. '$\Gamma \vdash \varphi$' means that the formula φ is derivable from the set of formulae Γ (and the axioms as stated below) using the above rules. '$\vdash \varphi$' is short for '$\emptyset \vdash \varphi$' and means that φ is a theorem, i.e., derivable from axioms alone without any additional assumptions. We describe derivability (i.e. proofs) below in Section 3.4.

Necessitation is sometimes called by other names, e.g., belief generalization. We have given it the name that reflects its origins in alethic modal logic [Che80]. It applies only to theorems of the logic. Like the Belief Conjuncatenation rule of BAN, this is often misunderstood and misapplied or improperly criticized. If using the assumptions about a protocol it is possible to derive that Q *said* X, it does not follow that P *believes* Q *said* X. This is because Q *said* X is merely derivable given the context of this protocol: it is not a theorem, i.e., derivable using logic alone.

Axioms of the logic are all instances of tautologies of classical propositional calculus [Men87], and all instances of the following axiom schemata.[4]

3.3 SVO Axioms

Belief Axioms

1. $(P \textit{ believes } \varphi \wedge P \textit{ believes } (\varphi \rightarrow \psi)) \rightarrow P \textit{ believes } \psi$
2. $P \textit{ believes } \varphi \rightarrow \varphi$

[3] This notation is motivated by that of [WK96] and is closer to that than to the notation in [SvO94,SvO96].

[4] Some of the following are proper axioms, logically. Those containing metavariables for formulae are actually axiom schemata. We will generally ignore this distinction, referring to all as 'axioms'.

3. P *believes* $\varphi \rightarrow P$ *believes* $(P$ *believes* $\varphi)$
4. $\neg(P$ *believes* $\varphi) \rightarrow P$ *believes* $(\neg P$ *believes* $\varphi)$

These are classic axioms of modal logic [Che80,Gol92] that were thoroughly analyzed in the early and middle part of the last century. (The classic names for them are **K**, **T**, **4**, and **5** respectively.) As in AT, belief is removed from most other axioms, the logic of belief is separated off from the rest of the logic. Readers familiar with modal logic will recognize these as the axioms[5] of the Lewis system **S5** [Che80]. This logical system is usually taken to characterize knowledge rather than belief, so we provide a brief side discussion of the point. It may be skipped without loss of continuity.

Discussion: Knowledge and Belief

Roger Needham has remarked in conversation that perhaps the biggest mistake they made with BAN was calling it a belief logic. We have already seen in the discussion of Nonce Verification in Section 2.2 that BAN-belief does not respect all of the intuitions of ordinary belief. Of course no technical usage could. Intuitions can be helpful, but they can also lead us from the main task of providing a useful analysis of authentication and authentication protocols. More detailed discussion of knowledge and belief in authentication logics can be found in [Syv92,AT91]. Here we cover a few of the main points.

The main question in distinguishing knowledge from belief is: If a principal thinks that φ, is he always right? If the answer is “Yes”, then we are talking about knowledge. If the answer is “No”, then we are talking about belief. That is overly simplistic, and philosophical counterexamples are thoroughly plumbed in the literature (e.g., look up the Gettier Problem [Mos86]). Nonetheless, it will do for our practical intentions. This translates naturally into the main formal question: Is it a theorem of the logic that $(P$ *believes* $\varphi \rightarrow \varphi)$?

This is just the **T** axiom introduced above as axiom 2. Faced now with this technical distinction, we can ask the practical question of which do we want. Surprisingly, the answer for our purposes is “I don’t care.” The reason is that this has played little role in actual analyses. So, if we don’t care, how did we come to add it as an axiom?

The answer lies in the semantics of the logic. One of the goals of this logic was that it have an intuitively reasonable model of computation and semantics, and that it be sound with respect to them. Mark Tuttle has noted that we ought to build models not logics in trying to capture our notions of authenticated communication [Tut]. It turns out that in [SvO96], the semantics of the intentional operator is based on an equivalence relation. And, it is a classic result of modal logic, that this is characteristic of the logic **S5**. So, it simply falls out that the above axioms are all valid in our semantics. But, this is not an important practical distinction. In practice, only axioms 1 and 3 seem to play a role.

[5] The **4** axiom (our axiom 3) is actually derivable given the others and is included for tradition and for its intuitive significance.

Source Association Axioms

5. $(P \overset{k}{\longleftrightarrow} Q \wedge R\ \textit{received}\ \{X\ \textit{from}\ Q\}_k) \rightarrow (Q\ \textit{said}\ X \wedge Q\ \textit{has}\ X)$

This replaces the Message Meaning Rule of BAN. Note the absence of the '*believes*' operator, and that the axiom applies when any principal R receives $\{X\ \textit{from}\ Q\}_k$. The logical significance of '$P \overset{k}{\longleftrightarrow} Q$' is now isolated from that of *believes*. As in BAN, there is also a corresponding public-key axiom.

6. $(PK_\sigma(Q,k) \wedge R\ \textit{received}\ X \wedge SV(X,k,Y)) \rightarrow Q\ \textit{said}\ Y$

We introduce some new notation here. '$SV(X,k,Y)$' means that applying k to X confirms that X is the result of signing Y with a private cognate of k. Note that this axiom separates out key binding (what principal is bound to the key) from key correctness (what key verifies the signature).

Key Agreement Axioms

7. $(PK_\delta(P,k_P) \wedge PK_\delta(Q,k_Q)) \rightarrow P \overset{F_0(k_P,k_Q)}{\longleftrightarrow} Q$
8. $\varphi \equiv \varphi[F_0(k,k')/F_0(k',k)]$
   $F_0(k',k)$ implicitly names the (Diffie-Hellman) function that combines k' with k^{-1} to form a shared key.

These are the axioms that characterize Diffie-Hellman key agreement. As we mentioned in Section 2.5, this is an important component in widely used authenticated key establishment protocols. We give here a brief account of the mathematics of Diffie-Hellman. For more details consult a standard cryptography text [MvOV97,Sch96].

Protocol 3 (Diffie-Hellman)

1. Assume that Alice and Bob share
   – a large prime p
   – a generator g of the multiplicative group $\mathbb{Z}_p^*$ of integers *modulo* p.
2. Alice chooses large integer x and computes $X = g^x \bmod p$
3. Bob chooses large integer y and computes $Y = g^y \bmod p$
4. Message 1 $A \rightarrow B:\ X$
   Message 2 $B \rightarrow A:\ Y$
5. Alice sets $k_{AB} = X^y \bmod p = g^{xy} \bmod p\ \ (= g^{yx} \bmod p)$
6. Bob sets $k_{AB} = Y^x \bmod p = g^{yx} \bmod p\ \ (= g^{xy} \bmod p)$

Receiving Axioms

9. $P\ \textit{received}\ (X_1, \ldots X_n) \rightarrow P\ \textit{received}\ X_i$, for $i = 1, \ldots, n$

10. $(P\ \textit{received}\ \{X\}_{k^+} \wedge P\ \textit{has}\ k^-) \rightarrow P\ \textit{received}\ X$
    Here k^+ and $k-$ are used to abstractly represent cognate keys, whether for symmetric or asymmetric cryptography. In the symmetric case, $k^+ = k^- = k$. In the asymmetric case, k^+ is a public key and k^- the associated private key.
11. $(P\ \textit{received}\ \lfloor X \rfloor_k) \rightarrow P\ \textit{received}\ X$
    Principals are assumed to possess public keys (for convenience).

Possession Axioms

12. $P\ \textit{received}\ X \rightarrow P\ \textit{has}\ X$
13. $P\ \textit{has}\ (X_1, \ldots, X_n) \rightarrow P\ \textit{has}\ X_i$, for $i = 1, \ldots, n$.
14. $(P\ \textit{has}\ X_1 \wedge \ldots \wedge P\ \textit{has}\ X_n) \rightarrow P\ \textit{has}\ F(X_1, \ldots, X_n)$
    'F' is meta-notation for any function computable in practice by P, e.g., encryption with known keys. The meaning of "computable in practice" is intentionally not formally determined. It could be, e.g., polynomial-time computable but will be treated as a black box, just as "encryption", "signature", etc., are in nearly all formal treatments of cryptographic protocols.

Comprehension Axiom

15. $P\ \textit{believes}\ (P\ \textit{has}\ F(X)) \rightarrow P\ \textit{believes}\ (P\ \textit{has}\ X)$
    'F' is meta-notation for any function that is effectively one-one (e.g., collision free hashes) and such that F^+ or F^- is computable in practice by P. Note that this axiom does not imply that F is invertible by P. Note also that the converse of this axiom is a derivable theorem.

Saying Axioms

16. $P\ \textit{said}\ (X_1, \ldots, X_n) \rightarrow P\ \textit{said}\ X_i \wedge P\ \textit{has}\ X_i$, for $i = 1, \ldots, n$.
17. $P\ \textit{says}\ (X_1, \ldots, X_n) \rightarrow (P\ \textit{said}\ (X_1, \ldots, X_n) \wedge P\ \textit{says}\ X_i)$, for $i = 1, \ldots, n$.

Freshness Axioms

18. $\textit{fresh}(X_i) \rightarrow \textit{fresh}(X_1, \ldots, X_n)$, for $i = 1, \ldots, n$.
19. $\textit{fresh}(X_1, \ldots, X_n) \rightarrow \textit{fresh}\ F(X_1, \ldots, X_n)$
    F must genuinely depend on all component arguments. This means that it is infeasible to compute value of F without value of all the X_i.

Jurisdiction and Nonce-Verification Axioms

20. $(P\ \textit{controls}\ \varphi \wedge P\ \textit{says}\ \varphi) \rightarrow \varphi$
21. $(\textit{fresh}(X) \wedge P\ \textit{said}\ X) \rightarrow P\ \textit{says}\ X$

Note: Neither axiom refers to belief

BAN Analysis	**SVO Analysis**
1. Idealize the protocol.	
2. Write assumptions about initial state.	a. Write assumptions about initial state.
3. Annotate protocol. For each message "$P \rightarrow Q : M$" of the idealized protocol, assert "Q *received* M".	b. Annotate protocol. For each message "$P \rightarrow Q : M$" of the (not idealized) protocol, assert "Q *received* M".
	c. Assert comprehensions of received messages.
	d. Assert interpretations of comprehended messages.
4. Use the logic to derive the beliefs held by protocol principals.	e. Use the logic to derive beliefs held by protocol principals.

Fig. 1. Protocol Analysis Steps

Symmetric Goodness Axiom

22. $P \overset{k}{\longleftrightarrow} Q \equiv Q \overset{k}{\longleftrightarrow} P$

3.4 Protocol Analysis

We now demonstrate how to do a protocol analysis using SVO. We will continue with our running example of the Needham-Schroeder Shared-Key Protocol 1. A comparison between BAN and SVO analysis is summarized in Figure 1.

NSSK Initial State Assumptions

As with BAN, the first assumptions to set out are the initial state assumptions. Unlike BAN, we do not idealize the protocol first. The assumptions are virtually the same as the initial state assumptions set out in the above BAN analysis, except that the jurisdiction assumption P5 is about jurisdiction over freshness of keys rather than jurisdiction over freshness of assertions about the goodness of keys.

P1. A *believes* $A \overset{k_{AS}}{\longleftrightarrow} S$
P2. B *believes* $B \overset{k_{BS}}{\longleftrightarrow} S$

P3. A *believes* S *controls* $A \overset{k}{\longleftrightarrow} B$
P4. B *believes* S *controls* $A \overset{k}{\longleftrightarrow} B$
P5. A *believes* S *controls fresh*(k)

P6. A *believes fresh*(n_A)
P7. B *believes fresh*(n_B)

P8. B *believes fresh*$(A \overset{k_{AB}}{\longleftrightarrow} B)$

NSSK Received Message Assumptions

This step is the same as the annotation assumptions in BAN, except that we here use the specified protocol, not its idealization. Amongst other things, this means that plaintext is not eliminated, and these premises can be read directly from the specification. These premises are not typically used directly in derivations. Rather, they are used in the production of comprehension premises, which are themselves used in producing interpretation premises.

P9. S *received* (A, B, n_A)
P10. A *received* $\{n_A, B, k_{AB}, \{k_{AB}, A\}_{k_{BS}}\}_{k_{AS}}$
P11. B *received* $\{k_{AB}, A\}_{k_{BS}}$
P12. A *received* $\{n_B\}_{k_{AB}}$
P13. B *received* $\{n_B - 1\}_{k_{AB}}$

NSSK Comprehension Assumptions

In this step, we express that which a principal comprehends of a received message. The move from the received message assumptions is usually straightforward in practice. In principle, this can be formalized. But a rigorous formalization makes for a very complicated logic and some of the intuitiveness of SVO is lost. Nonetheless, it may be desirable if the intent is to automate as much of the reasoning as possible. (Instances of such a formalization can be found in [WK96,Dek00].)

P14. S *believes* S *received* $(A, B, \langle n_A \rangle_{*s})$
P15. A *believes* A *received* $\{n_A, B, \langle k_{AB} \rangle_{*A}, \langle \{k_{AB}, A\}_{k_{BS}} \rangle_{*A}\}_{k_{AS}}$
P16. B *believes* B *received* $\{\langle k_{AB} \rangle_{*B}, A\}_{k_{BS}}$
P17. A *believes* A *received* $\{\langle n_B \rangle_{*A}$ *from* $B\}_{\langle k_{AB} \rangle_{*A}}$
P18. B *believes* B *received* $\{n_B - 1\}_{\langle k_{AB} \rangle_{*B}}$

NSSK Interpretation Assumptions

These assumptions are essentially the replacement for idealization. Producing them is inherently an informal process. We are asserting how a principal interprets a received message (as that principal understands it). This is inherently dependent on the protocol design. Idealization is one of the most criticized and/or misapplied aspects of BAN analysis—bad initial state assumptions being the other. While some informality seems necessary in anything like this framework, SVO analysis reduces the potential for problems. First, idealization is split into comprehension and interpretation. Second, and perhaps more important, the interpretational part of the process occurs after annotation rather than before. In idealization, there is a natural and correct tendency to interpret message components using formulae expressing the intent of the sender. BAN annotation then asserts that the receiver receives the intended meaning of the sender. By placing interpretation after annotation and comprehension, the focus naturally shifts to how the intent of the sender is understood by the receiver. That is, focus shifts from the meaning the sender had intended to the meaning that the receiver attaches to a received message.

P19. A *believes* A *received* $\{n_A, B, \langle k_{AB} \rangle_{*A}, \langle \{k_{AB}, A\}_{k_{BS}} \rangle_{*A}\}_{k_{AS}} \rightarrow$
A *believes*A *received*$\{n_A, B, A \xleftrightarrow{\langle k_{AB} \rangle_{*A}} B$, *fresh*$(\langle k_{AB} \rangle_{*A}), \langle \{k_{AB}, A\}_{k_{BS}} \rangle_{*A}\}_{k_{AS}}$

P20. B *believes* B *received* $\{\langle k_{AB}\rangle_{*B}, A\}_{k_{BS}} \to$

B *believes* B *received* $\{A \overset{\langle k_{AB}\rangle_{*B}}{\longleftrightarrow} B, \mathit{fresh}(\langle k_{AB}\rangle_{*B}), A\}_{k_{BS}}$

P21. $(A$ *believes* A *received* $\{\langle n_B\rangle_{*A}\}_{\langle k_{AB}\rangle_{*A}}) \wedge (A$ *believes* $A \overset{\langle k_{AB}\rangle_{*A}}{\longleftrightarrow} B) \to$

A *believes* A *received* $\{\langle n_B\rangle_{*A}, A \overset{\langle k_{AB}\rangle_{*A}}{\longleftrightarrow} B\}_{\langle k_{AB}\rangle_{*A}}$

P22. $(B$ *believes* B *received* $\{n_B - 1\}_{\langle k_{AB}\rangle_{*B}} \wedge (B$ *believes* $A \overset{\langle k_{AB}\rangle_{*B}}{\longleftrightarrow} B) \to$

B *believes* B *received* $\{n_B - 1, A \overset{\langle k_{AB}\rangle_{*B}}{\longleftrightarrow} B\}_{\langle k_{AB}\rangle_{*B}}$

Another point to note about these premises is that they all have conditional form. Often, the conditional is only a formal reminder that the interpretation depends on the comprehension of the actual receipt of the message. But in some cases, e.g., assumption P21, the interpretation depends not just on receipt of a message but also on other things, such as assumptions about good keys.

We have omitted an interpretation premise for the first message because it will play no role in the derivations. (The BAN assumption that plaintext does not affect analysis is not merely capricious.)

NSSK Derivations for Alice

1. $A believes A received \{n_A, B, A \overset{\langle k_{AB}\rangle_{*A}}{\longleftrightarrow} B, \mathit{fresh}(\langle k_{AB}\rangle_{*A}), \langle\{k_{AB}, A\}_{k_{BS}}\rangle_{*A}\}_{k_{AS}}$
   By Modus Ponens using P19, P15

Unlike in BAN, to move from here to Alice believing that the server sent the message she received requires several steps. We would have to apply the necessitation rule to the source association axiom, and also make use of propositional reasoning, axiom 1, modus ponens, etc. We will present here only the highlights, focusing on the authentication reasoning. We will generally omit reference to the rules (modus ponens and necessitation) and will refer to propositional reasoning or reasoning using the belief axioms by saying "by Belief Axioms".

2. A *believes* S *said* $\{n_A, B, A \overset{\langle k_{AB}\rangle_{*A}}{\longleftrightarrow} B, \mathit{fresh}(\langle k_{AB}\rangle_{*A}), \langle\{k_{AB}, A\}_{k_{BS}}\rangle_{*A}\}_{k_{AS}}$
   By Source Association, 1, P1, and Belief Axioms
3. A *believes* S *says* $\{n_A, B, A \overset{\langle k_{AB}\rangle_{*A}}{\longleftrightarrow} B, \mathit{fresh}(\langle k_{AB}\rangle_{*A}), \langle\{k_{AB}, A\}_{k_{BS}}\rangle_{*A}\}_{k_{AS}}$
   By Freshness, Nonce Verification, 2, P6, and Belief Axioms
4. A *believes* $A \overset{\langle k_{AB}\rangle_{*A}}{\longleftrightarrow} B$
   By Saying, Jurisdiction, 3, P3, and Belief Axioms
5. A *believes* $\mathit{fresh}(\langle k_{AB}\rangle_{*A})$
   By Saying, Jurisdiction, 3, P5, and Belief Axioms
6. A *believes* B *said* $(\langle n_B\rangle_{*A}, A \overset{\langle k_{AB}\rangle_{*A}}{\longleftrightarrow} B)$
   By Source Association, P21, 4, and Belief Axioms
7. A *believes* B *has* $\langle k_{AB}\rangle_{*A}$
   By Source Association, P21, 4, and Belief Axioms

8. A *believes* B *says* $(\langle n_B \rangle_{*A}, A \overset{\langle k_{AB} \rangle_{*A}}{\longleftrightarrow} B)$
By Freshness, Nonce Verification, 5, 6, and Belief Axioms

We could obtain similar results for Bob (assuming we use the dubious assumption P8, the dubiousity of which is discussed in Section 2.3). This concludes our analysis of the Needham-Schroeder Shared-Key protocol. As noted above, one of the motivations of SVO was to incorporate reasoning about Diffie-Hellman style key agreement. Analyses of two such protocols can be found in [SvO96].

3.5 The Nessett Protocol in SVO

What about the Nessett Protocol? How does it fare in SVO? Since SVO contains both negation and the ability to express possession, it can express who should not get keys. This was Nessett's primary concern about BAN.

More precisely. In SVO, we can state the requirement

$$\neg(E \textit{ has } k_{AB})$$

where 'E' is the adversary. Now, given our assumption of a Dolev-Yao adversary, it is perfectly reasonable, for every message M of every protocol to add to the annotation assumptions that E *received* M. It then becomes trivial for the Nessett protocol to derive

$$E \textit{ has } k_{AB}$$

So, we can prove Nessett Protocol to be insecure. But, what if we could not prove E *has* k_{AB}? What if this were merely consistent with the protocol not provable from it? As has been observed, failed proofs sometimes reveal attacks. But sometimes they simply reveal our inability to produce a proof. An independent semantics would allow us to evaluate the truth of assumptions and requirements. SVO was given, indeed based on, such a semantics. We defer discussion of it to Section 6. Another question that arises from this discussion is: just what are the goals of an authentication protocol? We now turn to this question.

4 Authentication Goals

In the previous sections, we have described the syntax of BAN logic [BAN89a] and its descendents, most notably SVO [SvO96], and demonstrated how their axioms and inference rules can be used to derive new information from the factual knowledge specifying a given protocol. For example, this allowed us in Section 2 to construct a formal argument in support of the idea that, by the end of a run of the Needham-Schroeder Shared-Key protocol, the involved parties believe that they share a good key for secure mutual communication. As a matter of fact, this was one of the intended functionalities of this protocol. In general, we will be particularly interested in those derivations that relate the specification of a protocol to its intended goals or requirements.

In the present section, we will be concerned with identifying the authentication goals that a given protocol may be expected to fulfill. Rather than directly attacking this general problem, we will trace the historical development of this quest. We start in Section 4.1 by examining which requirements can be expressed in BAN and then, in Section 4.2, we outline the contributions made by its successors, most notably VO [vO93]. Limitations in these approaches triggered the study of authentication goals per se, independently from their expressibility in any given specification language. Replays, i.e. unwanted behaviors due to the interferences of multiple runs of a protocol, were soon identified as a major cause of unsatisfiable authentication goals, which opened the door to subtle attacks. We examine them in Section 4.3. One of the results that emerged from this study is that specification languages such as BAN and SVO are not expressive enough to capture some of the authentication goals aimed at avoiding replays. The study of the notion of authentication continued, sometimes to tragi-comic extremes for most of the 1990's. We conclude in Section 4.4 by listing some of the problems that emerged and the proposed solutions.

4.1 BAN Authentication Goals

We have already outlined the way BAN goes about establishing authentication properties: given assumptions about the beliefs held by a principal before running a given protocol, it allows deducing beliefs that this principal must hold at the end of a run. This formal derivation is guided by the idealized protocol, which enriches the original specification with explicit descriptions of the intended functionality of selected message components, e.g. that a key is fresh or is good for communication between two principals. Although idealization is an essentially manual process and the logical status of the resulting annotations is dubious, the end-product is the vehicle that allows mapping what a principal believes before running a protocol to what he believes afterward, as described by the following diagram.

$$\begin{tikzcd} & \text{Idealized protocol} \arrow[d] & \\ \text{Pre-run beliefs} \arrow[r] & \boxed{\vdash_{\text{BAN}}} \arrow[r] & \text{Post-run beliefs} \end{tikzcd}$$

In the case of the Needham-Schroeder Shared-Key protocol examined in Section 2, from assumptions such as that principal A possesses a good key to communicate with a server S ("A *believes* $A \xleftrightarrow{k_{AS}} S$" in symbols) and that the nonce n_A is fresh ("A *believes fresh*(n_A)"), we deduced that she can legitimately think that she is handed a good key k_{AB} to communicate with a receiver B (i.e. "A *believes* $A \xleftrightarrow{k_{AB}} B$"). The derivation relies on protocol idealizations such as "$A \xleftrightarrow{k_{AB}} B$". We can indeed instantiate the above schema in the following partial diagram:

$$\begin{tikzcd}
 & \cdots, A \overset{k_{AB}}{\longleftrightarrow} B, \cdots \arrow[d] & \\
\begin{array}{l} A \textit{ believes } A \overset{k_{AS}}{\longleftrightarrow} S \\ A \textit{ believes fresh}(n_A) \end{array} \arrow[r] & \boxed{\vdash_{\mathrm{BAN}}} \arrow[r] & A \textit{ believes } A \overset{k_{AB}}{\longleftrightarrow} B
\end{tikzcd}$$

BAN logic does not define the notion of authentication. Instead, it offers means to express the fact that certain properties, clearly related to authentication, should be valid at the end of a message exchange, assuming certain premises hold. We call them *authentication goals*, and use the phrasing *authentication assumptions* for the premises they depend upon. BAN logic views authentication as a protocol dependent notion: therefore, different protocols will generally require different authentication assumptions and achieve different authentication goals. We schematically describe BAN's approach to authentication in the following diagram, a final evolution of pictures above:

$$\begin{tikzcd}
 & \text{Idealization of protocol } \mathcal{P} \arrow[d] & \\
\begin{array}{r} \text{Authentication} \\ \text{assumptions for } \mathcal{P} \end{array} \arrow[r] & \boxed{\vdash_{\mathrm{BAN}}} \arrow[r] & \begin{array}{l} \text{Authentication} \\ \text{goals for } \mathcal{P} \end{array}
\end{tikzcd}$$

The realization that authentication is a protocol dependent notion leads to our first *observation on authentication*:

1. *There is not a unique definition of authentication that all secure protocols satisfy.*

Although authentication goals depend on the protocol at hands (and on its assumptions), certain goals recur fairly often. In particular, all BAN analyses of key distribution protocols have some of the following formulae as conclusions:

– A *believes* $A \overset{k_{AB}}{\longleftrightarrow} B$
– B *believes* $A \overset{k_{AB}}{\longleftrightarrow} B$
– A *believes* B *believes* $A \overset{k_{AB}}{\longleftrightarrow} B$
– B *believes* A *believes* $A \overset{k_{AB}}{\longleftrightarrow} B$

Clearly, other goals are possible, e.g. about beliefs concerning public keys. But they do not arise in the examples considered in [BAN89a], in which public keys are always a means to produce session keys in the form of shared keys, rather than the objects about which one would try to establish goals.

On the other hand, there are good key distribution protocols[6] for which some of the above goals are not applicable. Consider for example, the Wide-Mouthed Frog Protocol [BAN89a,CJ97], described below:

[6] See the end of Section 4.4 below.

Protocol 4 (Wide-Mouthed Frog) [BAN89a,CJ97]

Message 1 $A \rightarrow S: \ \{T_A, B, k_{AB}\}_{k_{AS}}$
Message 2 $S \rightarrow B: \ \{T_S, A, k_{AB}\}_{k_{BS}}$

The initiator A wants to communicate securely with another party B. She achieves this by generating a key k_{AB} and sending it to a trusted server S together with her intention to communicate with B. The server simply forwards this key to B, together with the identity of the generator. The components T_A and T_S are timestamps. The first message is encrypted with a key k_{AS} that A shares with S, which ensures that its contents are not accessible to any other party. Similarly, the second message is encrypted with key k_{BS} that B shares with S. By the end of a run, only A, B and S are expected to know k_{AB}.

Since A generates the key k_{AB}, she has jurisdiction over its freshness and intended use. Therefore

$$A \textit{ believes } A \overset{k_{AB}}{\longleftrightarrow} B$$

is an assumption as well as a goal for the protocol. We leave it to the interested reader to devise the other relevant assumptions of the Wide-Mouthed Frog Protocol 4 as well as the form of its idealization. Provable goals of this protocol include

$$B \textit{ believes } A \overset{k_{AB}}{\longleftrightarrow} B \qquad \text{and} \qquad B \textit{ believes } A \textit{ believes } A \overset{k_{AB}}{\longleftrightarrow} B$$

(again, the formal derivation is left to the reader). It should however be observed that the formula

$$A \textit{ believes } B \textit{ believes } A \overset{k_{AB}}{\longleftrightarrow} B$$

although taken from the above list of typical goals, is not provable. Indeed A cannot hold any belief about B's beliefs since she is not the recipient of any message in this protocol.

4.2 VO Authentication Goals

As reported in Section 2, BAN was shown to have a number of shortcomings, soon to be fixed by a succession of proposals. While early extensions such as GNY [GNY90] and AT [AT91] concentrated on providing a finer modeling language for protocol actions, the logic VO [vO93] also enriched the lexicon of formally stated authentication goals. At the same time, it exposed nuances of the still vague notion of authentication that BAN and its early successors were unable to express. Altogether, VO provided a better understanding of what authentication actually is.

In this section, we present the various forms of authentication available in VO. Rather than trying to be completely faithful to [vO93], we will incorporate some of the adjustments made in later proposals. For simplicity, we will formalize the notions in this section using the syntax of SVO [SvO96], already introduced in Section 3.

We first need to introduce some syntax. VO expresses the fact that a key k is an *unconfirmed secret* for a principal P to communicate with another principal Q as

$$P \overset{k-}{\longleftrightarrow} Q$$

Similarly to BAN's key goodness, this expression means that only P and Q (and third parties trusted by both) know k. It also implies that P has access to this key (e.g. by having received it in a message), but does not enforce a similar requirement on Q: this principal may not be aware of k. It was later observed [SvO96] that this expression can be given the following simple definition:

$$P \overset{k-}{\longleftrightarrow} Q \quad \equiv \quad (P \overset{k}{\longleftrightarrow} Q \wedge P \textit{ has } k)^7$$

It should be observed that key confirmation is not symmetric. Indeed, $P \overset{k-}{\longleftrightarrow} Q$ is not equivalent to $Q \overset{k-}{\longleftrightarrow} P$.

Given this definition, VO distinguishes the following six forms of authentication:

Ping authentication captures situations where a principal P wants to know whether an interlocutor Q is alive. It is expressed as the following formula, where X can be any message.

$$P \textit{ believes } Q \textit{ says } X$$

Observe that not only should Q have uttered something (X), but he should have done so recently, as enforced by the use of *says* as opposed to *said*.

Entity authentication further requires that P's interlocutor Q said something relevant to their present conversation. Given some information Y_P known to be fresh to P (e.g. a nonce), entity authentication mandates that Q recently sent a message $F(X, Y_P)$ from which it is manifest that Q has seen Y_P and has processed it. This is captured by the following formula:

$$P \textit{ believes } (Q \textit{ says } F(X, Y_P) \wedge \textit{fresh}(Y_P))$$

Suitable message transformation functions F must possess the following properties:

– F is effectively one-to-one, by which we mean that for any choice of the arguments X and Y, if $F(X, Y) = Z$, it is computationally infeasible to find values X' and Y' different from X and Y such that $F(X', Y') = Z$. As in [MvOV97], p. 324, the meaning of 'computationally infeasible' is "intentionally left without formal definition", to be interpreted relative to an understood frame of reference. For example it might mean that

7 In BAN, a principal A could only refer to a key k by believing some property of it, most notably its goodness, or by receiving it in a message. GNY [GNY90] remedied to this deficiency by allowing one to talk about entities possessed by a principal. Here we adopt the AT syntax "*A has k*" introduced in Section 3.

there is no algorithm that terminates in a time polynomial in the size of the argument of F that computes such X' and Y'. But this is only one possibility.
This definition also indicates that F genuinely depends on Y_P, in the sense that it is computationally infeasible for an adversary to produce an alteration Y'_P of Y_P that yields the same result, even if he controls the choice of the first argument of F.

- F is computable in practice by Q. This too is left without precise definition. One example would be, given X and Y_P, Q can compute $F(X, Y_P)$ in polynomial time.
- P can effectively verify that the received value $F(X, Y_P)$ has actually been constructed by using Y_P. This can be achieved in two different ways:
  - P can in practice compute enough of the inverse of F to expose the use of Y_P. This is the case, for example, if $F(X, Y_P) = \{Y_P\}_X$ and X is P's public key.
  - P has access to X (and Y_P), can effectively compute $F(X, Y_P)$, and can verify whether the result corresponds to the value transmitted by Q. An example of this situation is when F is a hash function and X is known to P.

Secure key establishment indicates that a principal P believes that he has a good key k to communicate with a counterpart Q. Given the above notion of unconfirmed secret, this goal is easily expressed by the following formula:

$$P \textit{ believes } P \overset{k-}{\longleftrightarrow} Q$$

Key freshness simply requires that a principal P believes a key k to be fresh:

$$P \textit{ believes fresh}(k)$$

Mutual understanding of shared keys applies to situations where a principal P can establish that an interlocutor Q has sent a key k as an unconfirmed secret between the two of them (from Q's point of view). This is formalized by the following formula:

$$P \textit{ believes } Q \textit{ says } (Q \overset{k-}{\longleftrightarrow} P)$$

Key Confirmation is intended to describe scenarios in which a principal P believes that an interlocutor Q has proved to have received and successfully processed a previously unconfirmed secret key k between the two of them. Similarly to the case of entity authentication, we capture the "confirmation" aspect of this definition by requiring Q to return k to P, modulo the application of a function F that is effectively one-to-one, computable in practice by Q, and effectively verifiable by P. We have the following formal definition:

$$P \textit{ believes } (P \overset{k-}{\longleftrightarrow} Q \wedge Q \textit{ says } F(k))$$

It should be observed that key confirmation is not the same as the BAN-style second-order belief "P *believes* Q *believes* $P \overset{k}{\longleftrightarrow} Q$", which may wrongly imply that Q believes that k is a good key for them to communicate. For a similar reason, it differs from mutual understanding of shared keys "P *believes* Q *says* $Q \overset{k-}{\longleftrightarrow} P$".

These definition shed substantial light on the notion of authentication. However, they also raise further questions, a clear indication that VO has moved our understanding of authentication forward, but also that it has not exhausted the subject. A closer look at entity authentication and mutual understanding of shared keys will reveal some problems, that will be addressed in the rest of this section.

Given actual principals A and B, the intended meaning of the entity authentication goal

$$A \textit{ believes } (B \textit{ says } F(X, Y_A) \wedge \textit{fresh}(Y_A))$$

is that A is engaged in a protocol run with B and she thinks that B said something in response to the nonce Y_A she generated for this run. Observe however that this goal does not impose any constraint on B's assumptions; an intruder could indeed have rerouted messages in such a way that Y_A entered a conversation B was having with a third principal, C say; B may then have freshly sent $F(X, Y_A)$ to C, but the intruder altered the intended course of this message so that it reached A instead. This undesirable behavior passes the above entity authentication test.

Consider now the following goal, an instance of the mutual understanding of shared keys:

$$A \textit{ believes } B \textit{ says } (B \overset{k-}{\longleftrightarrow} A)$$

The concern raised above for entity authentication does not apply here since the presence of the unconfirmed secret expression $B \overset{k-}{\longleftrightarrow} A$ indicates that B is aware of the fact that k is intended to communicate with A. There is however room for attacks: again, an intruder may have rerouted messages so that B thinks that the key k is being used in a run r he is conducting with A, while A believes she is using it in a different run r', although with B. Again, this potentially harmful behavior satisfies the above notion of mutual understanding of shared keys goal.

Both scenarios arise as (intruder-assisted) misunderstandings: the involved principals are participating in an apparently legal run of the protocol, but not in the same run and not necessarily with each other. Both situations involve an interleaving of at least two protocol runs, with the intruder altering the message routes to unduly connect these otherwise independent runs. Such situations are called replays and will be examined in detail in the next section.

4.3 Replay Attacks

As we just discussed, a *replay attack* is characterized by an intruder opportunistically bending the path of messages belonging to different runs of a protocol, possibly after making minor changes to the messages themselves. An in-depth study

A. Run external attacks

1. Interleavings
   (a) Deflections
      i. Reflections
      ii. Deflections to a third party
   (b) Straight replays
2. Classic Replays
   (a) Deflections
      i. Reflections
      ii. Deflections to a third party
   (b) Straight replays

B. Run internal attacks

(a) Deflections
   i. Reflections
   ii. Deflections to a third party
(b) Straight replays

Fig. 2. A Full Taxonomy of Replays [Syv94]

of the different incarnations of the notion of replay was undertaken in [Syv94]. We present this analysis and use it to measure the expressiveness of the authentication logics from Sections 2 and 3. Two attempts at covering more replay attacks, one that refines the BAN model of time and one that introduces the notion of role, are then discussed.

A Taxonomy of Replays

Syverson, in [Syv94], proposes two orthogonal classifications of replays, which formalize the observations that these misbehaviors derive from the interleaving of multiple protocol runs, and that the intruder redirects messages among them, respectively. We will now examine them in detail. The overall combined taxonomy is displayed in Figure 2.

A first way to approach replay attacks is to distinguish them on the basis of which runs the replayed messages are taken from. This materializes in a *run taxonomy* [Syv94], which immediately branches into the following two classes:

A. In a *run external attack*, the replayed message comes from outside the current protocol run. This option involves the execution of at least two runs, which can be either concurrent or sequential, as indicated by the next branching point in this taxonomy:
   (1) An *interleaving attack* requires two or more runs to take place contemporaneously. The intruder uses the different runs in turn as oracles to answer the challenges set forth by the others. A popular example of this form of replays is given by Lowe's attack [Low96] on the Needham-Schroeder Public-Key Authentication Protocol 7, which we will examine in Section 4.4.
   (2) An attack that involves external runs but without the requirement that they should be contemporaneous is called a *classic replay*. The intruder remembers messages sent back and forth during previous runs, and opportunistically replays them to mount an attack on the current run. We

have seen an example of classic replay in Section 2 as the Denning-Sacco attack [DS81] on the Needham-Schroeder Shared-Key Authentication Protocol 1.

B. An attack can also result from opportunistically replaying messages from the current protocol run. These are known as *run internal attacks*. An example involving the Neuman-Stubblebine repeated authentication protocol [NS93,CJ97] has been exposed by Syverson in [Syv93b] and by Carlsen in [Car93].

Another way to look at replay attacks examines which messages are rerouted by the intruder, and how this is done. The resulting classification is known as the *destination taxonomy* [Syv94]. Let us first consider who the replayed message was intended for:

a. The first situation, called *deflection*, redirects the replayed message to a principal different from its intended recipient. This situation can be further refined in the following subcases:
   (i) First, the replayed message can be sent back to its sender. This is called a *reflection attack*.
   (ii) We can also have a *deflection to a third party*, in which the message in question is redirected to a principal that is neither the intended recipient or the originator.

b. An intruder can mount an attack by channeling a message to its intended destination, but with some delay and possibly in a different run of the protocols. This is known as a *straight replay*.

We will now demonstrate the various forms of destination attacks by examining a well-known disruption on the a variant of a draft protocol due to Yahalom, a version of which was ultimately published in [Yah93]. The variant we consider here was first presented in [BAN89a]. By virtue of this iterated genesis, we call it the BAN-Yahalom protocol. It is specified as follows:

Protocol 5 (BAN-Yahalom) [BAN89a]

Message 1	$A \to B:$	A, n_A
Message 2	$B \to S:$	$B, n_B, \{A, n_A\}_{k_{BS}}$
Message 3	$S \to A:$	$n_B, \{B, k_{AB}, n_A\}_{k_{AS}}, \{A, k_{AB}, n_B\}_{k_{BS}}$
Message 4	$A \to B:$	$\{A, k_{AB}, n_B\}_{k_{BS}}, \{n_B\}_{k_{AB}}$

The initiator A and the responder B rely on a server S to generate a key k_{AB} that would allow them to communicate securely. The long term keys k_{AS} and k_{BS} guarantee the mutual authentication of the server and the principals A and B, respectively. Intentionally, the third message indirectly authenticates B to A by having the server encapsulate both B's identity and A's fresh nonce n_A

in the message $\{B, k_{AB}, n_A\}_{k_{AS}}$. The fourth message authenticates A to B by encrypting B's nonce n_B with the newly acquired (and supposedly secure) key k_{AB}.

This protocol is subject to the following attack, first presented in [Syv94], which makes use of three protocol runs, which we distinguish by using different numerals and indentations. The intruder is given the name E (the Eavesdropper), and we write E_P to indicate an action of the attacker while impersonating principal P. The attack unfolds as follows:

$$\begin{array}{llllll}
1. & & & A & \to E_B: & A, n_A \\
 & I. & & E_B & \to A\ : & B, n_A \\
 & II. & & A & \to E_S: & A, n'_A, \{B, n_A\}_{k_{AS}} \\
 & & ii. & E_A & \to S\ : & A, n_A, \{B, n_A\}_{k_{AS}} \\
 & & iii. & S & \to E_B: & n_A, \{A, k_{AB}, n_A\}_{k_{BS}}, \{B, k_{AB}, n_A\}_{k_{AS}} \\
3. & & & E_S & \to A\ : & n_E, \{B, k_{AB}, n_A\}_{k_{AS}}, \{A, k_{AB}, n_A\}_{k_{BS}} \\
4. & & & A & \to E_B: & \{A, k_{AB}, n_A\}_{k_{BS}}, \{n_E\}_{k_{AB}}
\end{array}$$

In line (1), A generates the nonce n_A to communicate with B. The outgoing message is intercepted by E and replayed to A in line (I) after altering its postulated originator to B. In A's view, this is the first message of a different run, with B as its originator, and therefore she responds as expected by generating a nonce n'_A and forwarding the message $(A, n'_A, \{B, n_A\}_{k_{AS}})$ to the server in line (II). The intruder alters this message *en route* by replacing the nonce n'_A with n_A in line (ii). Logically this is part of a third run of the protocol (the server has no reason to suspect that this run lacks its first message). The server performs its task on line (iii) by generating the message $(n_A, \{A, k_{AB}, n_A\}_{k_{BS}}, \{B, k_{AB}, n_A\}_{k_{AS}})$. These two inner runs are left dangling. We return instead to the outer run, where A is expecting a reply from S to her indirect request of line (1) via B. In line (3), the intruder replays the message captured in line (iii) after substituting a nonce n_E of his own in place of the outermost occurrence of n_A. This message has the expected form, and therefore A replies in line (4) as dictated by the text of the protocol.

Although no key is revealed to the intruder E, an attack has taken place since A believes she has been talking to B without this principal even participating in any run. This is clearly a failure of authentication. In order to mount the attack, the intruder makes use of three replay techniques:

- Going from lines (1) to (I), we first have a reflection of the nonce n_A back to A.
- Going from lines (II) to (ii), we have a straight replay of the message components A and $\{B, n_A\}_{k_{AS}}$ across two different runs of the protocol.
- Finally, going from lines (iii) to (3), we have a third party deflection of the encrypted components $\{A, k_{AB}, n_A\}_{k_{BS}}$ and $\{B, k_{AB}, n_A\}_{k_{AS}}$ from S to A and away from B.

Figure 2 integrates the run and destination taxonomies of replays, showing in this way all possibilities for a replay attack. This is therefore a complete classification.

Gauging Expressiveness

The above taxonomy of replays gives a clear view of the different ways an intruder can take advantage of the messages exchanged in one or more runs of a protocol to mount an authentication attack. This minute classification is also an excellent basis to measure the expressive power of various protocol analysis formalisms: an ideal system would successfully apply to all points in Figure 2. Most proposals cover instead a more spotty spectrum. In this section, we will make use of this taxonomy to outline the strengths and weaknesses of the authentication logics discussed in Sections 2 and 3. The results of this analysis should be taken with a grain of salt: there are cases where a formalism does not have mechanisms to systematically expose a certain class of attacks and yet has tackled specific instances of this class.

We shall first consider BAN logic [BAN89a] introduced in Section 2. Freshness is the only mechanism available in BAN to distinguish a run from another. This is a very weak mechanism indeed, since its effect is limited to temporally partitioning protocol actions into recent (i.e. provably fresh) and old. Therefore, freshness alone cannot hope to reveal run internal attacks, nor any form of interleaving attack. It instead focuses on the portion of the run taxonomy [Syv94] that we have called classic replays.

Although by assumption rather than by analysis, BAN captures a similarly small fragment of the destination taxonomy [Syv94]. First, recall that BAN expects a principal to recognize messages he/she has said. This is equivalent to limiting the scope of the verification process to protocols that are immune to reflection attacks. Second, the notion of a (shared) key k being good for two principals A and B to communicate, '$A \overset{k}{\longleftrightarrow} B$', similarly circumvents deflection-to-third-party attacks. What is left of the destination taxonomy is the category of straight replays and some deflection-to-third-party situations that involve public keys.

In summary, the expressiveness of BAN relative to Figure 2 is limited to the zones marked "straight replays", and the area pertaining to "classic replays" among the run external attacks. In spite of this restricted scope, BAN has been successfully used to perform a large number of analyses.

The logic GNY [GNY90] corrects the inability of BAN to talk about reflection attacks by providing syntax (an asterisk "$*$") to flag a message as "not originated here". The other limitations of BAN remain. Surprisingly they are not addressed by the successors of GNY, namely AT [AT91], VO [vO93], and SVO [SvO96].

We can sum up these observations as follows: none of the discussed logics exhausts or fully expresses the notion of authentication. In particular, since all of them, starting with BAN, are equipped to reason about freshness, we deduce that, in general, authentication problems cannot be reduced to enquiries about freshness. This leads to our second observation on authentication:

$$\boxed{\text{2. } \textit{Freshness is not rich enough to express all the kinds of authentication.}}$$

Adding Time to Increase Expressiveness

As mentioned above, BAN and its successors rely on a simplicistic view of time that only distinguishes "recent" events from "old" actions. Freshness declarations draws the temporal line separating them, although recent messages almost always pertain to the current run. A finer use of time in protocol analysis was proposed in [Syv93a] with the introduction of the modality $\diamondminus$, read "previously". This allows not only breaking the time-line in more than two segments, but also expressing the fact that event occurrences should have happened according to a certain order. For example, a requirement such as

$$A \textit{ received } \{B, k_{AB}, n_A\}_{k_{AS}} \rightarrow \diamondminus(B \textit{ said } \{A, n_A\}_{k_{BS}})$$

means that if A receives the message on the left-hand side of the implication, then B has previously sent the message on the right-hand side. The added temporal operator has therefore the additional effect of capturing a form of causality between events.

A natural question to ask is whether the addition of the above modality to BAN or SVO is sufficient to address all forms of replay in the taxonomy in Figure 2. The answer is unfortunately negative: at least straight replays are not covered.

In order to demonstrate this point, we will rely on the protocol below, first presented in [Sne91]. The system it models consists of a master computer M and a collection of sensors $S_1, \ldots, S_n$, each controlled by a microprocessor. The master computer periodically queries the sensors. The protocol is aimed at authenticating the order and timeliness of their reports.

Protocol 6 (Snekkenes) [Sne91]

Message 1 $M \rightarrow S_i : \mathsf{Query}(i, j)$

Message 2 $S_i \rightarrow M : \lfloor n_{ij}, \mathsf{Query}(i, j), \mathsf{Answer}(i, j) \rfloor_{k_i^{-1}}$

Message 3 $M \rightarrow S_i : \lfloor n_{ij} \rfloor_{k_M^{-1}}$

In the first message, M sends a query $\mathsf{Query}(i, j)$ to sensor S_i, where j is a progressive number. In the second message, the invoked sensor, S_i returns an answer $\mathsf{Answer}(i, j)$ together with the original query and a nonce n_{ij} aimed at ensuring the freshness of the reply. The origin of this composite message is guaranteed by having S_i sign it with its private key k_i^{-1}. Upon receiving this message, M responds by signing the nonce n_{ij} with his own key k_M^{-1}.

This protocol is not immune to straight replay attacks, even if M keeps track of all used n_{ij} and (correctly) assumes these values are fresh. An intruder can indeed subvert the result of this protocol by intercepting a query $\mathsf{Query}(i, j)$ on its way from M to S_i, forwarding it multiple times to S_i, and letting through to M the most desirable answer.

This attack can be neutralized by reversing the order of the last two messages of this protocol. Consequently, the nonce n_{ij} is now generated by M rather than by S_i. Moreover, it is now the sensor's duty to memorize the nonces, verify their freshness, and limits its answers to one per nonce, to preclude replays. The master computer shall maintain an association between nonces and queries to prevent the subversive rerouting of signed nonces to sensors different from the one they were intended for.

Snekkenes observed in [Sne91] the rather unsettling fact that the BAN analysis of both variants of this protocol is same. He furthermore proved that a similar limitation holds for any two variants of a given protocol that differ only by the order of the exchanged messages:

Theorem 1. *(Snekkenes '91)*

1. *Let $\mathcal{P}$ and $\mathcal{P}'$ be protocols composed of the same messages, although not necessarily in the same order.*
2. *Assume that $\mathcal{P}$ can be shown to satisfy some goal G given certain assumptions $\mathcal{A}$.*
3. *Furthermore, assume that $\mathcal{P}'$ is demonstrably insecure.*

Then, $\mathcal{P}'$ can also be shown to satisfy the goal G given the assumptions $\mathcal{A}$.

This result was rigorously proved in the context of an annotated sequent calculus for BAN logic.

The above theorem states that extending BAN queries to faithfully account for the causal ordering of protocol actions is not sufficient to prevent all forms of replay attacks. This leads to our third observations on authentication:

3. *Correct causal order and source of a message are not strong enough for all authentications.*

Roles in Cryptographic Protocols

The most visible effect of the introduction of the temporal operator $\diamondminus$ in the previous section was to extend the language used to express and validate protocol requirement. A similar proposal in [Bie90] focused instead on the language used to specify a protocol.

In [Bie90], protocols are described in the logic of knowledge and time CKT5, which enriches a fragment of first-order logic with the modal operator $K_{A,t}$ and a suitable set of axioms. The intended meaning of a formula of the form $K_{A,t}\,\varphi$ is that at time t principal A knows that φ holds. The use of proper quantifiers over time variables allows capturing the relative temporal ordering of events, similarly to what we have observed with $\diamondminus$.

The introduction of this modality makes it possible to put strict temporal constraints on the actions that a principal participating in a protocol is allowed to perform. This permits expressing scenarios where, for example, if A sent m_1 and A received m_2, then the next action of A is to send m_3. In this proposal, the protocol actions available to a principal are organized in a *role*, given as the sequences of message transmissions that this principal is going to perform, possibly in response to the reception of some well-defined messages. A protocol specification is then presented as a set of roles, one for each participating principal. It should be observed that this approach constitutes a radical change of course with respect to the BAN-like specification methodology discussed so far: in these languages, a protocol was described by listing the messages exchanged during an expected run, while roles focus on the individual view of each principal, independently from any run.

The CTK5 specifications given in [Bie90] allowed each honest principal participating in a protocol to play exactly one role. It was shown in [Sne92] that this restriction could give an incorrectly clean bill of health: attacks that relied on having the same principal act both as an initiator and a responder, for example, were missed. This same paper corrected this limitation by upgrading the one-to-one relation between roles and principal proposed in [Bie90] to a many-to-one correspondence. Therefore, a given principal was now associated with a set of roles, an entity also known as a *multi-role*. Differently from roles, multi-roles could, for example, express the necessary conditions to set up the attack on the BAN-Yahalom protocol discussed in Section 4.3.

The CKT5 formalization of roles and multi-roles used in [Sne92] was later simplified in [Car94], which also gave an algorithm to generate CKT5 role specifications from the BAN-like "standard notation" of a protocol.

Clearly, if the protocol at hand is constrained in such a way that every honest principal can play at most one role, then no multi-role flaws can be uncovered. Even in this limited setting, the use of CTK5 as a specification language does not prevent the possibility of all attack. The Snekkenes Protocol 6 from Section 4.3 is subject to the same attack even when expressed in this language. We can therefore strengthen our last observation on authentication as follows:

4. *Correct causal order and message source, and freedom from multi-role flaws are not strong enough for all authentications.*

4.4 A Child's Garden of Authentications

Starting with the most common authentication objectives of BAN logic [BAN89a], the previous section has described the contributions made by various researchers to the formalization and understanding of the notion of authentication. We saw how these original goals were extended in languages such as VO [vO93] and SVO [SvO96]. We then categorized attacks relative to the taxonomy of replays defined in [Syv94] and finally discussed a series of proposal aimed at repairing specification [Bie90,Sne92,Car94] and requirement [Syv93a] deficiencies of the BAN family of logics. Yet, not all attacks could be nailed down.

By this time, we were in the mid 1990s and the notion of authentication was looking like a more and more distant chimera. The research toward this holy grail intensified, and considerable effort was spent trying to answer the following basic question:

Is there an adequately strong criterion for freedom from replay?

In this section, we will report on some of the progresses that were made toward this elusive goal. We shall anticipate that this question is still open. As we will see in Section 5, this quest is not however as popular as it once was, mainly because several researchers have now given guidelines aimed at constructing protocols that are free from attacks by design.

What do we Mean by Entity Authentication?

Gollmann raised the question in the title of this section in the homonymous paper [Gol96]. The notion of *entity authentication* had been used liberally, often abused, in the security literature (we gave one of the many definitions in Section 4.2). Gollmann's paper discusses various meanings attributed to this phrase, and crystallizes some of these definitions in the context they ought to be used.

One of the strongest meanings of "entity authentication" requires that all the communications that constitute a session be accessible only to the involved parties, or to some entity in whose integrity they can put a reasonable amount of confidence. This degree of authentication is usually attained by encrypting all the communications between two principals by means of a *session key* freshly generated in a secure manner by a trusted third party. This constitutes the essence of Gollmann's first authentication goal:

G1: The protocol establishes a fresh session key, known only to the session parties and possibly to a trusted server.

While this goal is sufficient when considering protocol runs in isolation, situations that may involve several runs require reinforcing this requirement with the following clause:

G1': Furthermore, compromising old sessions keys does not lead to the compromise of new session keys.

In particular, new session keys should not be transmitted encrypted with old session keys.

A second meaning of "entity authentication" requires that a principal A can ascertain that an interlocutor B has received and successfully interpreted a message sent by A to B. Gollmann expresses this requirement as follows, modulo minor editing:

G2: A key associated with a principal B was used in a message received by another principal A in the protocol run, in a response to a challenge issued by A in the form of a nonce or a timestamp.

This is what we called "entity authentication" in Section 4.2.

A yet weaker form of "entity authentication" simply requires a principal to be able to ascertain that an intended interlocutor was active during a protocol run. This is expressed as the following goal:

G3: A key associated with a principal B was used during the protocol run, in a response to a challenge issued by another principal A in the form of a nonce or a timestamp. However, A did not need to receive a message where this key was used.

This is essentially what we called "ping authentication" in Section 4.2.

Agreements

In [Low97], Lowe observed that all definitions used to talk about authentication have the following form:

> A protocol $\mathcal{P}$ guarantees property X to initiator A for another principal B,
>
> *iff*
>
> whenever A completes a run of the protocol, apparently with responder B, then a certain requirement ψ holds.

We denote the condition "whenever A completes a run of the protocol, apparently with responder B" as φ_{AB}. Then, all the definitions can be seen as implications of the following form:

$$\varphi_{AB} \rightarrow \psi.$$

Here, A and B are parameters rather than specific principals. Therefore, although these goals may appear to be bound to the principals, they are actually more general.

It should be observed that these goals are validated once a run is completed. Therefore, they are intended to authenticate runs, rather than individual messages as in the case of the requirements for BAN examined in Section 4.1.

We will now examine some of the property-requirement pairs (X, ψ) considered in [Low97]. These definitions refine and give a more precise meaning to notions such as ping or entity authentications discussed above.

Aliveness: $\psi =$ "B has been running the protocol".

This requirement extends ping authentication to protocol runs. When satisfied, it guarantees that A's interlocutor, B, has been active some time in the past. Situations in which the run proceeds smoothly from A's point of view without B taking part in any action represent a failure of aliveness. We have observed such a situation in the attack to the BAN-Yahalom Protocol 5 in Section 4.3.

Like every requirement discussed in [Low97], there is a *recent* version of aliveness: $\psi =$ "B has been running the protocol *recently*". Recent aliveness requires B to have been active during the current run. Notice that B does not need to have been running the same protocol as A, and even if he did he may have run it with a different party.

It should be noted that recent aliveness is not only stronger than ping authentication, but it also subsumes VO's entity authentication discussed in Section 4.2. Indeed, recent aliveness is manifested in a run of a cryptographic protocol by witnessing precisely the transformations required by this form of authentication. From that point of view, recent aliveness possibly gives a meaning to the notion of entity authentication.

Weak agreement: $\psi =$ "B has previously been running the protocol, *apparently with A*".
Weak agreement strengthens aliveness by requiring not only that A's interlocutor B was active, but that A had evidence that he participated in a very direct manner by decrypting or signing messages that he only could have processed (unless the relevant keys were compromised). Observe that weak agreement does not require B to be running the protocol with A, nor can it he assumed to don the expected role (e.g. if A acts as the initiator, B may not necessarily be playing the responder role).

In [Low96,Low97], the difference between (recent) aliveness and weak agreement was illustrated by the attack below, which has achieved world fame and has become a major test bed, sometimes even a rite of passage, for every new protocol verification tool. Lowe's attack operates on the following fragment of a protocol due to Needham and Schroeder [NS78], the public-key version of the Needham-Schroeder Shared-Key Protocol 1 analyzed in Section 2.3.

Protocol 7 (Abridged Needham-Schroeder Public-Key) [NS78]

Message 1	$A \to B:$	$\{n_A, A\}_{k_B}$
Message 2	$B \to A:$	$\{n_A, n_B\}_{k_A}$
Message 3	$A \to B:$	$\{n_B\}_{k_B}$

In this protocol, the initiator A sends her identity and a freshly generated nonce n_A to B, protecting this message by encrypting it with B's public key k_B. Upon receiving it, B generates a nonce of his own, n_B, and sends it to A together with n_A, encrypted with A's public key. In the last message, A sends n_B back to B, encoded with k_B. The protocol originally described in [BAN89a] had an initial key distribution phase in which A and B requested and received the keys k_A and k_B from a trusted server.

Upon completing a run of this protocol, A can be confident that she has been talking with B. Lowe's attack [Low96] shows that this protocol does not provide the reverse assurance. The trace of this attack is as follows:

$$\begin{array}{rlcll}
1. & A & \to & E & \{n_A, A\}_{k_E} \\
i. & E_A & \to & B & \{n_A, A\}_{k_B} \\
ii. & B & \to & E_A & \{n_B, n_A\}_{k_A} \\
2. & E & \to & A & \{n_B, n_A\}_{k_A} \\
3. & A & \to & E & \{n_B\}_{k_E} \\
iii. & E_A & \to & B & \{n_B\}_{k_B}
\end{array}$$

On line (1), A starts the protocol with the intruder E, who accesses the contents of the first message, re-encrypts it with B's public key and forwards it to this principal in line (i). On line (ii), B replies as if the message had come directly from A. The attacker intercepts it and directs it to A in lines (ii) and (2). The initiator A completes the protocol with E by encrypting the nonce n_B she received with E's public key. Finally, E forwards this nonce encrypted with k_B to B. In the end, A (correctly) believes that she has been running the protocol with E, but B is fooled into assuming that he has been talking to A.

It is clear that this attack proves that the Needham-Schroeder Public-Key protocol does not satisfy weak agreement from B's point of view (i.e. after swapping A and B in the above definition). However, it is proved in [Low97] that this protocol satisfies (recent) aliveness.

This attack is routinely used in courses and lectures to support the idea that protocol analysis is difficult, and in seminars and papers to motivate new proposals in this area. It is indeed true that it revealed a novel vulnerability to a protocol published 18 years earlier, and proved correct by a number methods, most notably using the BAN logic [BAN89a]. However, the Needham-Schroeder Public-Key Protocol was never deployed in any real-life setting. More importantly, a careful reading of [NS78] indicates that Lowe's weak agreement for the responder was not among the goals of this protocol.

Among the authors who challenged the legitimacy of this attack, Gollmann [Gol00] observed that it does not reveal any flaw if B's objective in this protocol was to have a communication with A. This corresponds to the notion of ping authentication (Gollmann calls it "authenticating packets"). However, if this protocol was used to establish a secure channel between the two parties, then Lowe's attack is a clear manifestation of a violation. Gollmann called this situation "authenticating circuits".

Non-injective agreement (with respect to data set ds)**:** ψ = "B has previously been running the protocol, apparently with A, *and B was acting as the responder in his run, and both agree on values of variables in ds*".
A yet stronger form of authentication is given by non-injective agreement. Here, A's interlocutor, B, is required to play the expected role, and their runs need to be synchronized to the extent that their respective variables among ds contain the same values. Observe however that this goal does not guaranteed a one-to-one relationship between A's and B's runs (hence the name).
The Needham-Schroeder Public-Key Protocol 7 does clearly not satisfy this requirement. There are however protocols that pass the weak agreement test, but fail non-injective agreement. Examples include the Andrew Secure RPC Handshake [Sat89,CJ97], and Snekkenes Protocol 6 analyzed in Section 4.3.

Agreement: This goal, sometimes called *injective* agreement, reinforces non-injective agreement with the requirement that there is a *one-to-one correspondence between runs*. This last goal in [Low97] forces the runs of each involved party to by fully synchronized, and therefore may appear as the ultimate authentication requirement.
The Wide-Mouthed Frog Protocol 4 presented in Section 4.1 can be proved to satisfy non-injective agreement, but does not pass the stronger agreement

test. An attack that exemplifies this situation is presented in [CJ97]: the adversary replays the server's message to B within the lifetime of the timestamp, essentially acquiring a new timestamp from the server, and repeats this game until A tries to run a legitimate session of the protocol with B, at which point he can replay the appropriate message to B. This attack can be formally expressed in a logic that includes time. A formal analysis in CSP using PVS is given in [ES00].

Intensional Specification

Lowe's hierarchy of authentication goals discussed in the previous section was essentially a response to *intentional specification*, a perhaps overly strict notion of protocol correctness defined by Roscoe in [Ros96]. The definition of intentional specification is as follows:

> A party cannot believe that a run has completed successfully unless a series of messages that agree on all parameters has occurred, up to and including the last message communicated by the given party.

In [Low97], Lowe observes that intensional specification is such a strong requirement that only the most inconsequential behaviors could violated it and yet satisfy agreement. Examples of such failures of intensional specification are:

– Assume that, in response to a request, a server sends a pair of messages (m_A, m_B) to principal A. This party can decrypt m_A, but not m_B, and is expected to forward this component to another principal B, who is able to interpret it. We have seen an instance of this scenario in the BAN-Yahalom Protocol 5 discussed in Section 4.3. Lowe's first example of an intensional specification "attack" that passes the agreement test relies on an adversary that substitutes m_B with some random value X in the message from the server to A. Then, it reinstalls m_B in place of X in the second message from A to B.

– Lowe's second "attack" example takes place in a situation where a server sends messages m_A and m_B to principals A and B, respectively, and in that precise order. An intruder delays the first message so that m_B reaches B before m_A reaches A.

It has been debated whether these failures can reasonably be seen as attacks, in any even remotely practical meaning of the term. In particular, it is not clear whether there are "real" attacks that satisfy agreement but not intensional specification. These doubts are highlighted by analyzing the following previously unpublished protocol.

Protocol 8 (Unpublished)

Message 1	$A \rightarrow B:$	$\{n_A, A, B\}_{k_{AB}}$
Message 2	$B \rightarrow A:$	$\{n_B, n_A, A, B\}_{k_{AB}}$
Message 3	$A \rightarrow C:$	$\{A, C, (n'_A \oplus n_B)\}_{k_{AC}}$
Message 4	$C \rightarrow A:$	$\{n_C, A, C, (n'_A \oplus n_B)\}_{k_{AC}}$
Message 5	$A \rightarrow B:$	$\{n_B, (n''_A \oplus n_C), A, B\}_{k_{AB}}$

Principals A and B set up a mutual challenge involving nonces n_A and n_B in line (1) and (2). In line (3) and (4), a similar process occurs between A and C, but the fresh value in the third message is not properly a nonce, but the result of taking the X-OR of B's nonce n_B from line (2) and some newly generated nonce n'_A. In the last message, A answers B's challenge from line (2), but also includes one of these pseudo-nonces, which is obtained by taking the X-OR of C's nonce n_C and yet another nonce n''_A generated by A.

Although we did not conduct a formal proof, this protocol seems to satisfy Lowe's notion of agreement. There are however situations in which it violates Roscoe's intentional specification:

– Suppose that A sends the message in line (3) before receiving the nonce n_B in line (2): she could for example use $n'_A = n^*_A$ to form this message without taking any X-OR. While the one-to-one correspondence between runs is not affected, intensional specification is violated since C would receive the nonce n'_A rather than the pseudo-nonce $(n'_A \oplus n_B)$. This may be potentially harmful since the causal relation of messages appears to be affected.
– Suppose now that A generates a nonce n^*_A before receiving B's nonce in line (2), waits for n_B, and only then calculates $n'_A = n^*_A \oplus n_B$ and sends the message in line (3). Now intensional specification seems to be satisfied. However, the end-result is identical since, being X-OR associative and idempotent, $(n^*_A \oplus n_B) \oplus n_B = n^*_A$. Indeed, the value sent to C has been decided before receiving B's message.
– Last, consider an identical scenario, but in which A generates n^*_A *after* receiving B's message in line (2), but without using n_B. Now, the causal relation between the messages is clearly respected, yet C will receive a value that is independent from the nonce n_B.

Similar "attacks" can be constructed with respect the the messages on lines (4) and (5).

Matching Histories

Matching histories [DvOW92] is an older proposal whose strength fits between Lowe's (injective) agreement and weak agreement. This characterization of authentication is particularly interesting because its definition was developed by industrial specialists in secure system design and cryptography rather than by formal methods experts, as for the proposals discussed so far. In particular, their focus was likely to be on a more practical articulation of the notion of "authentication" geared toward actual applications rather than on mapping out the theoretical terrain.

A protocol satisfies matching histories if the following condition can be proved to hold:

> When a principal A accepts the other party's identity (before receiving or sending further messages), the other party's record of the partial or full run matches A's (with the same values for all message variables).

This requirement is as strong as Lowe's (injective) agreement insofar as the number of runs and all variables must match between A and B. It is however not as powerful as weak agreement since B does not need to have been running the protocol with A. It can however be shown that matching histories and agreement are equivalent if every message exchanged in a protocol includes the identities of the apparent sender and of the intended recipient.

It is interesting to observe that matching histories is motivationally similar to VO's key confirmation (see Section 4.2),

$$P \textit{ believes } (P \overset{k-}{\longleftrightarrow} Q \wedge Q \textit{ says } F(k))$$

while Lowe's various "agreements" goals are motivationally similar to VO's mutual understanding of shared keys (see Section 4.2) and to BAN's second-order belief (see Section 4.1),

$$P \textit{ believes } Q \textit{ says } (Q \overset{k-}{\longleftrightarrow} P) \qquad\qquad P \textit{ believes } Q \textit{ believes } (Q \overset{k}{\longleftrightarrow} P).$$

Cautionary Note

By the end of the 1990s, the research on issues related to authentication had proliferated to the point that some practitioners started noticing a dichotomy between the problems addressed in the academic literature on security, and the solutions sought in real world scenarios. Gollmann, again, voiced these concerns in the paper [Gol00]. He observed that the research in this area was often fueled by a perceived informality in protocol analysis, and, putting it in his own words,

> *[this] motivates the presentation of a new formalism for the analysis of authentication protocols, and the biggest prize to be won is the detection of an attack hitherto unreported. We will argue that such exercises in formal analysis more often add to the problem than help in its resolution.*

Furthermore:

> *Perceived problems with authentication are caused by intuitive but imprecise interpretations of the objective of "authentication", and by neglecting to take into account the environment a protocol is intended to operate in. In many cases, new attacks do not expose subtle flaws in protocols but differences in assumption about protocol goals.*

However, sometimes they do expose subtle flaws. Furthermore, new theories sometimes do turn out to be practically useful. Clearly, this is not always the case, but even then, they often have an impact on our understanding of the various concepts that contribute to what we call security. In these cases (and many other), it is essential not to mistake theoretical results for applied ones, or vice versa.

5 Design Principles and Protocol Logics

At this point we take a brief holiday from formal characterizations of authentication to consider protocols from a more informal and more applied perspective.

5.1 Protocol Design Principles

Abadi and Needham set out "prudent engineering practices for cryptographic protocols" in [AN94,AN96]. These are rules of thumb for good protocol design. They are not meant to apply to every protocol in every instance, but they do provide a laundry list of things that should be considered when designing a protocol. The paper contains useful examples and discussion of the principles. We quote from [AN96] just the principles here and then briefly comment on them below.

PRINCIPLE 1. Every message should say what it means: The interpretation of the message should depend only on its content. It should be possible to write down a straightforward English sentence describing the content—though if there is a suitable formalism available, that is good too.

PRINCIPLE 2. The conditions for a message to be acted upon should be clearly set out so that someone reviewing the design may see whether they are acceptable or not.

PRINCIPLE 3. If the identity of a principal is essential to the meaning of a message, it is prudent to mention the principal's name explicitly in the message.

PRINCIPLE 4. Be clear as to why encryption is being done. Encryption is not wholly cheap, and not asking precisely why it is being done can lead to redundancy. Encryption is not synonymous with security, and its improper use can lead to errors.

PRINCIPLE 5. When a principal signs material that has already been encrypted, it should not be inferred that the principal knows the content of the message. On the other hand, it is proper to infer that the principal that signs a message and then encrypts it for privacy knows the content of the message.

PRINCIPLE 6. Be clear about what properties you are assuming about nonces. What may do for ensuring temporal succession may not do for ensuring association—and perhaps association is best established by other means.

PRINCIPLE 7. The use of a predictable quantity (such as the value of a counter) can serve in guaranteeing newness, through a challenge-response exchange. But if a predictable quantity is to be effective, it should be protected so that an intruder cannot simulate a challenge and later replay a response.

PRINCIPLE 8. If timestamps are used as freshness guarantees by reference to absolute time, then the difference between local clocks at various machines must be much less than the allowable age of a message deemed to be valid. Furthermore, the time maintenance mechanism everywhere becomes part of the trusted computing base.

PRINCIPLE 9. A key may have been used recently, for example to encrypt a nonce, yet be quite old, and possibly compromised. Recent use does not make the key look any better than it would otherwise.

PRINCIPLE 10. If an encoding is used to present the meaning of a message, then it should be possible to tell which encoding is being used. In the common case where the encoding is protocol dependent, it should be possible to deduce that the message belongs to this protocol, and in fact to a particular run of the protocol, and to know its number in the protocol.

PRINCIPLE 11. The protocol designer should know which trust relations his protocol depends on, and why the dependence is necessary. The reasons for particular trust relations being acceptable should be explicit though they will be founded on judgment and policy rather than on logic.

5.2 Design Principle comments

Such rules of thumb should always be considered when designing a protocol and only violated when the violation is consciously done for a superseding reason. Since the rules are generally quite compelling, we focus on some of the ways in which they might not apply, as a caution against applying them blindly. (Comments in this section are mostly drawn from [Syv96].)

Building on the above principles, Anderson and Needham set out further principles specifically focused on public-key protocols. Their first principle is an expansion of PRINCIPLE 5 above.

> Sign before encrypting. If a signature is affixed to encrypted data, then one cannot assume that the signer has any knowledge of the data. A third party certainly cannot assume that the signature is authentic, so non-repudiation[8] is lost. ([AN95], p. 237, Principle 1)

This is a nice principle for illustrating limitations: there are many places where non-repudiation may not be of paramount concern; thus the principle may be too narrow. For example, anonymity may take priority over non-repudiation. This would occur in voting protocols, and in digital cash. Digital cash often makes use of a blind signature, in which the authority issuing the cash signs a 'coin' that has been 'blinded' so that the authority cannot recognize the specific coin and thus tie it to the principal to whom it was issued. After signing the blinded coin, the principal unblinds it so that anyone can recognize it as a coin authentically signed by the issuer.[9]

[8] The goal of non-repudiation is to prevent a principal from denying some action s/he has taken, such as sending or receiving a message.

[9] This is a very simple description. Blinding was invented by Chaum [Cha83]. More on digital cash and other applications of blinding can be found in [Sch96].

This principle may also be too broad: signing encrypted data may be necessary for non-repudiation. One place this can be seen is in a coin-flip protocol. A principal signs encryptions of "Heads" and "Tails" and later reveals the encryption key. Part of the reason is so that she cannot deny the choices offered and also so that the opposing principal cannot deny the choice made. A simple coin-flip protocol protocol demonstrating this point was given in [Syv96]. What follows is an even more simple version of this (without, e.g., replay protection). Other similar protocols were discussed in [Tou92].

Protocol 9 (Simple Coin Flip)

Message 1 $A \rightarrow B$: $\lfloor \{Heads\}_k, \{Tails\}_k \rfloor_{k_A^{-1}}$
Message 2 $B \rightarrow A$: $\lfloor X \rfloor_{k_B^{-1}}$ (where X is one of $\{Heads\}_k$ or $\{Tails\}_k$)
Message 3 $A \rightarrow B$: $\lfloor k \rfloor_{k_A^{-1}}$
Message 4 $B \rightarrow A$: $\lfloor k \rfloor_{k_B^{-1}}$

Non-repudiation is a fairly subtle requirement. It may be unsurprising that principles such as the one under discussion are subject to the cautionary remarks we have been making. Explicitness, however, would seem to be paramount in all security protocols, and especially in authentication protocols. Indeed, Abadi and Needham regard it (as embodied in PRINCIPLES 1 and 2 above) as the overarching principle in the design of secure cryptographic protocols. It is therefore surprising that there are authenticated key distribution protocols that can only function in the absence of explicitness (especially explicitness as in PRINCIPLE 10). We now present such a protocol.

Protocol 10 (EKE — Encrypted Key Exchange) [BM92,BM93]

Message 1 $A \rightarrow B$: $A, \{k_A\}_P$
Message 2 $B \rightarrow A$: $\{\{k_{AB}\}_{k_A}\}_P$
Message 3 $A \rightarrow B$: $\{n_A\}_{k_{AB}}$
Message 4 $B \rightarrow A$: $\{n_A, n_B\}_{k_{AB}}$
Message 5 $A \rightarrow B$: $\{n_B\}_{k_{AB}}$

The idea of the EKE protocol is to function as a privacy multiplier. Let Alice be some client and Bob a server for which Alice has password P. P is thus a secret shared between A and B, and the only means of authentication A possesses. She encrypts a public key k_A with P and sends it to Bob. Bob generates a session key k_{AB} and encrypts this with k_A and then encrypts the result with P. There is then a handshake that shows fresh possession of the session key. The important thing to observe about this protocol is that the content of messages cannot be confirmed upon receipt since the recipient of a message cannot tie its content to any known values until s/he completes the protocol. In particular, principals cannot tell if received messages have the correct form for them to take the next step. It is only when a recipient gets his last message that he can confirm that the preceding messages had the correct content and acting upon them was appropriate. If any of the messages contained adequate redundancy in content or coding for a principal to know what s/he is receiving (or sending) before the end of the protocol, then the protocol would be vulnerable to guessing attacks since P is a weak secret.

Even if this protocol is a counterexample to the complete generality of explicitness, it is also an example for another of the design principles; Anderson and Needham warn

> Be careful when signing or decrypting data that you never let yourself be used as an oracle by your opponent. ([AN95], p. 240, Principle 3)

EKE puts a spin on that principle; instead of preventing use of principals as oracles it ensures that the output of such oracles is of no use to the attacker.

Despite such unusual examples, explicitness is very often exactly what is required. We now delve deeper into its implications.

5.3 Fail-Stop Protocols

For any definition of authentication, almost all of the failures in the literature are due to active attacks in which a message is somehow altered or substituted for another in a way it was not intended. Thus, stopping such attacks would go a long way towards a general guarantee of protocol security. Fail-stop protocols [GS98] are designed to meet this goal.

Using Lamport's definition of causality [Lam78], we can organize the messages of a protocol into an acyclic directed graph where each arc represents a message and each directed path represents a sequence of messages. In a fail-stop protocol, if a message actually sent is in any way inconsistent with the protocol specification, then all those messages that come after this altered message on some path in the graph (i.e., they are causally after the altered message) will not be sent. Obviously conditions to act upon all protocol messages must be explicit in the content and format of each message in order for the protocol to be fail-stop.

A protocol is said to be *fail-stop* if any attack interfering with a message in one step will cause all causally-after message in the next step or later not to be sent [GS98].

No definition of authentication given so far is sufficient for fail-stop. The main reason is that the definitions we have discussed are focussed on properties that must hold if and when a principal has completed a protocol run. But, fail-stop is a requirement that must hold as the protocol executes. For example, consider the EKE protocol of the last section. This is quintessentially not fail-stop. A principal cannot confirm anything about the content or possibly even encoding of any message until s/he has received the last message of the protocol run.

Claim 1 *Active attacks cannot cause the release of secrets within the run of a fail-stop protocol.*

Claim 1 follows immediately from the definition of a fail-stop protocol, because active attacks do not cause more (or different) messages to be sent; so an attacker using active attacks cannot obtain more secrets than one using passive eavesdropping.

One of the desirable features of a fail-stop protocol is this form of immunity to active attacks. More generally, since an active attack will cause a fail-stop protocol to halt, in a fail-stop protocol no principal will ever produce encryptions or any other computations on data from a message that was not entirely legitimate. Therefore, we need to consider only passive attacks in which an adversary records messages and tries to compute secrets from them. Such passive attacks (and protection measures against them) are much better understood than active attacks and easier to analyze. And, as already noted, they are substantially less common in the attack literature.

This shows us the beginnings of a synergy between design principles and formal analysis, except that fail-stop is not quite a design principle. But the synergy can be strengthened via explicitness based on the principles of Abadi and Needham.

One of the ways to make a protocol fail-stop is to design it in accordance with the following criteria:

1. The content of each message has a header containing the identity of its sender, the identity of its intended recipient, the protocol identifier and its version number, a message sequence number, and a freshness identifier.
2. Each message is encrypted under a key shared between its sender and intended recipient.
3. An honest principal follows the protocol and ignores all unexpected messages.
4. A principal halts any protocol run in which an expected message does not arrive within a specified timeout period.

Here a freshness identifier can be a timestamp (if clocks are assumed to be securely and reliably synchronized) or a nonce issued by the intended recipient. But, the freshness identifier in the first message of the protocol cannot be a nonce since the recipient must be able to determine if the protocol should proceed based on it. So, it must be a sequence number, timestamp, or something that will meaningfully indicate freshness to the recipient. When a freshness identifier takes on a more complicated form, the rules for reasoning about freshness in sections 2 and 3 can be used to determine if the identifier is fresh with regard to the recipient. Basically, these rules say that, if x is deemed fresh and y cannot be computed (in a computationally feasible way) by someone without the knowledge of x, then y is also deemed fresh. Encryption with a shared key in item 2 of this claim can be replaced by the use of an encryption using the recipient's public key of a signature using the sender's private key. We can offer no formal proof of the claim, but it should be clear by inspection.

It might seem that fail-stop protocols automatically guarantee authentication.

Protocol 11 (Simple Fail-Stop Example)

Message 1	$A \rightarrow B:$	$\{A, B,\ \mathit{Prot\_name},\ \mathit{version},\ \mathit{seq.}{=}\,1,\ T_A,\ \mathit{Query}\}_{k_{AB}}$
Message 2	$B \rightarrow A:$	*Response.*

In the first message T_A is a timestamp, and other fields have their obvious meaning. This message clearly follows the format of above design criteria. The second message is not of that format, but since it is the last one in the protocol, there are no causally-after messages. Thus, the protocol is fail-stop. However, the second message is not authenticated (according to virtually any definition).

Extensible Fail-Stop Protocols

A protocol can be fail-stop even if it contains messages that could have come from any principal at any time. In this section we explore a strengthening of the fail-stop concept.

A message in a protocol is *last* if no protocol message is causally after it. A protocol is *extensible fail-stop* (EFS) if adding any last message to the protocol results in a fail-stop protocol.

Note that limiting to ”ping-pong” protocols (where each message is followed by a single successor) implies a unique last message. This is the typical case for two party authentication protocols. The example of Protocol 11 is not EFS because adding a another message after Message 2 would result in a protocol that is not fail-stop. For EFS protocols, authentication is in fact automatically guaranteed—but only message authentication. An example of an EFS protocol is as follows:

Protocol 12 (Simple EFS Example)

Message 1	$A \rightarrow S:$	$\{A,\ S,\ \mathit{Prot.},\ \mathit{vers.},\ \mathit{seq.}{=}\,1,\ T_1,\ \mathit{request}(A, B)\}_{k_{AS}}$
Message 2	$S \rightarrow A:$	$\{S,\ A,\ \mathit{Prot.},\ \mathit{vers.},\ \mathit{seq.}{=}\,2,\ T_2,\ (k, A, B)\}_{k_{AS}}$
Message 3	$A \rightarrow S:$	$\{A,\ S,\ \mathit{Prot.},\ \mathit{vers.},\ \mathit{seq.}{=}\,3,\ T_3,\ (k, A, B)\}_{k_{AS}}$
Message 4	$S \rightarrow B:$	$\{S,\ B,\ \mathit{Prot.},\ \mathit{vers.},\ \mathit{seq.}{=}\,4,\ T_4,\ (k, B)\}_{k_{BS}}$
Message 5	$B \rightarrow S:$	$\{B,\ S,\ \mathit{Prot.},\ \mathit{vers.},\ \mathit{seq.}{=}\,5,\ T_5,\ (k, B)\}_{k_{BS}}$

This example demonstrates that fail-stop, even extensible-fail-stop, does not imply that the protocol satisfies all kinds of authentication. In the example, all messages are authenticated, but Bob does not know with whom he shares a key. Even a protocol in which Bob is given the wrong name for the principal meant to share the key could still be EFS.

Roughly, most of the authentication properties discussed in Section 4 are properties established by a complete protocol run about messages and the content of messages sent during that run. But, (extensible) fail-stop properties are authentication properties established by messages about the complete protocol run. For example, the following claim is immediate.

Claim 2 *Extensible fail-stop protocols are immune to replay.*

There is another, potentially more interesting property of EFS protocols.

Claim 3 *The sequential and parallel composition of EFS protocols is extensible fail-stop.*

The claim is justified by cases. For parallel composition: a message inserted causally before a last message of a fail-stop protocol will be ignored or cause a halt. Thus, it will not cause an EFS protocol to cease to be EFS. For sequential composition: let Pr_1 and Pr_2 be two EFS protocols. Suppose that some or all of the messages of Pr_1 are received after a last message of PR_2. If the first message of Pr_1 causes the result to be non-EFS, then Pr_2 was not EFS. (Contradiction.) And, if any later message causes the result to be non-EFS, then Pr_1 was not EFS.

We have already seen that fail-stop protocols need only to be examined for secrecy in the context of passive attacks (because active attacks cannot cause the release of secrets). In addition to its inherent interest, Claim 3 provides another design advantage of EFS protocols. Even analyses that consider interleaving typically assume only one protocol is running. If protocols are EFS, we are free to run multiple protocols in one environment without concern for interleaving attacks.[10]

As noted above, EFS protocols can be simply designed using basic explicitness rules. EFS protocols more flexible wrt composability, and EFS rules simplify the analysis task by removing replay considerations. How else might design rules synergize with protocol analysis logics?

5.4 Design Rules and Protocol Logics

A straightforward way for design rules to synergize with protocol logics is to build design checks directly into the logic. Brackin did precisely that in [Bra00]: he designed the logic BGNY [Bra96], based on GNY. He later developed an associated automated HOL tool, AAPA (Automated Authentication Protocol Analyzer) [Bra98], and a specification language similar to Millen's CAPSL [DM00,Mil]. The resulting system appears to be easy to use. Brackin has analyzed the entire Clark-Jacob library[11] using AAPA. He has also analyzed large commercial protocols such as the Cybercash main sequence protocol [Bra97]. This alone makes his a significant body of work, although we are not primarily concerned with automated tools in this paper.

Wedel and Kessler devised another BAN logic we will call 'WK' [WK96]. WK works with an automated tool AUTLOG based on Prolog. One advantages of the WK approach is that no formulae occur in messages. This is another step in solving the problem of the informal nature of idealization. Of course there is still

[10] See Section 7 for a cautionary note.

[11] The Clark-Jacob library is a fairly comprehensive list of known attacks on published authentication protocols [CJ97].

the need for interpretation assumptions (as they were called in Section 3.4). That part cannot be automated. However, analysis in WK automates derivation of the comprehension assumptions. Recall that these were the assumptions that allowed us to express what a principal understands of received messages even though some of the message may be unfamiliar or not decryptable by the principal. In fact, the WK notation for not-understood messages motivated the notation given above in Section 3, although the use by Wedel and Kessler is not exactly the same. Another automated tool in the BAN family is the recent C3PO of Anthony Dekker [Dek00]. This is a GUI tool based on the Isabelle theorem-prover. The logic associated with this tool is called 'SVD', and, like WK, it is a variant on SVO. Neither of these has the published track record of analyses of Brackin's work, however.

A different approach to automation that again combines logics and design is that taken by Clark and Jacob in [CJ00]. In some sense the idea of this approach is to not do design at all. Rather goals are stated and then protocols are synthesized that meet these goals. Clark and Jacob automatically generate protocols from basic BAN logic goals (as described in Section 4.1) using genetic algorithms and simulated annealing. Another automated synthesis, but based on Song's Athena model checker rather than on BAN, was presented by Perrig and Song in [PS00]. Related ideas can be found in [Gut]. This no-design approach may have great long term potential, but it is still early. As we have seen, even simple protocols are subtle and the contribution of such approaches may be to produce protocols with desirable features that no person would be likely to design.

Buttyán, Staamann, and Wilhelm also synthesize protocols from a BAN-like logic [BSW98]. However, unlike the previously mentioned approaches that effectively generate random protocols and then prune to the results that meet desired goals, they directly synthesize protocol designs from goals. Their protocol designs are slightly more abstract than we have been considering. They specify and reason about protocols on the more abstract level of channels. The encryption mechanism used to secure the channel is regarded as an implementation issue. The result is thus somewhat similar to spi calculus [AG99] but is closer to Needham-Schroeder style specifications. Roughly speaking, their design logic synthesis rules work by running an abstracted version of BAN in reverse. For example, if C is a channel, then the following is a synthesis rule.

$$\frac{P\ \textit{believes}\ P\ \textit{received}\ X\ \textit{on}\ C}{P\ \textit{sees}\ X\ \textit{on}\ C \qquad P\ \textit{can read}\ C}$$

A protocol that would satisfy the goal above the line would need to have P receiving X on channel C, where C might be, e.g., encryption using P's public key or a key P shares with another principal.

These rules give articulated goals, not conclusions. In some cases they yield intermediate goals that require further applications of rules before the protocol can be synthesized.

A common theme of this design logic and the synthesis tools is that one first specifies what is wanted then looks at the protocol. That means that one must state generic requirements in a formal language. The intuitive expression of requirements is thus a strong advantage of the logical approach. This also suggests another way of combining formal requirements statements with existing formal analysis techniques: Give the semantics of requirements language in the language of the formal analysis method. Then, use the formal analysis method to evaluate the truth of requirements statements in models of the protocol.

6 Semantic Approaches

We have seen in the previous section that it is often advantageous to use distinct languages to express a protocol under investigation and the goals it is expected to meet. The protocol specification language typically has an operational flavor that makes it particularly adequate for analyses based on simulation, such as model checking. Requirements are more easily stated in declarative formalisms, preferably with strong logical foundations. In order to be usable, requirements need to be mapped down to the execution model supported by the protocol specification language. We do so by endowing the requirement logic with an operational semantics in terms of the formalism used to express the protocol.

In this section, we will briefly examine two instances of this symbiosis. First, in Section 6.1, we look at the successful NRL Protocol Analyzer [Mea94,Mea96] together with the NPATRL requirements logic [SM96]. Then, in Section 6.2, we discuss a recently proposed synergy that adopts the popular strand formalism [THG97,THG98b] as an operational model and a BAN-like logic as the specification formalism [Syv00].

6.1 NPATRL

Our first case study will consist of the established synergy between the NRL Protocol Analyzer [Mea94,Mea96] and the NPATRL requirement language [SM96]. We first sketch relevant aspects of the NRL Protocol Analyzer and then introduce NPATRL.

The NRL Protocol Analyzer Model

The NRL Protocol Analyzer, or NPA for short, is a computer-assisted verification tool for security protocols which combines model checking and theorem-proving techniques to establish authentication and secrecy properties. We will limit the presentation of this system to the aspects that will be relevant to our discussion of the NPATRL language. The interested reader is invited to consult [Mea94,Mea96] for further details.

A protocol is modeled as a number of communicating state machines, each associated with a different roles. Their transitions correspond to the actions that comprise the corresponding role. At run time, roles are executed by *honest principals* who faithfully follow the protocol. Several instances can be executing at the same time, and they are distinguished by means of a unique round number.

The intruder is modeled after the Dolev-Yao adversary, described in Section 1.1. *Dishonest principals* share their keys and other confidential information with the adversary.

The messages in transit, the information held by each principal and the intruder, the runs currently being executed, and the point that each of them has reached constitute the global *state* of the NRL Protocol Analyzer. A protocol action implements a local transformation with global effects on the state. The initial state is implicit in the protocol specification.

In order to verify a protocol, a specification is fed into the run-time system of the NRL Protocol Analyzer together with the description of a family of states that correspond to attack situations. The system applies protocol actions backwards from these target states until it either reaches the initial state, or it exhausts all possibilities for doing so. In the first case, it reports the sequence of transitions that link these two states: this tracks a possible attack. The second case establishes that an attacker cannot produce the target scenario. Although the search space is in general infinite, the NRL Protocol Analyzer incorporates techniques based on theorem proving that have the effect of soundly restricting the search to a finite abstraction, in most cases. We can pictorially describe the operations of the NRL Protocol Analyzer by means of the following diagram, where we have kept the fairly stable intruder model implicit:

$$\begin{tikzcd}
 & \begin{array}{c}\text{Family of}\\ \text{undesirable final states}\end{array} \arrow[d] & \\
\begin{array}{r}\text{NPA protocol}\\ \text{specification}\end{array} \arrow[r] & \boxed{\text{NPA checker}} \arrow[r] & \text{OK / Attack}
\end{tikzcd}$$

As it regresses back towards the initial state, the NRL Protocol Analyzer maintains a *trace* of the sequence of actions that, when executed, lead to the target state. If the initial state is ever reached, the sequence constructed in this manner is returned as a description of the attack it has found. When a path is abandoned, the corresponding trace fragment is discarded. Traces are sequences of *events* of the following form:

$$\mathit{event}(P, Q, T, L, N)$$

In general, any protocol or intruder state transition may be assigned an event. The arguments are interpreted as follows: P is the principal executing the transition, Q is the set of the other parties involved in it, T is a name that identifies the transition, L is a set of relevant words, and N is the local round number of the transition. There are three categories of events which correspond to receiving a message (predicate "receive"), accepting data as valid as a result of performing certain checks (predicate "accept"), and sending a message (predicate "send"). Here are two examples:

$$\text{accept}(\text{user}(A, \text{honest}), [\text{user}(B, H)], \text{initiator\_accept\_key}, [K], N)$$

$$\text{send}(\text{server}, [\text{user}(A, \text{honest}), \text{user}(B, \text{honest})], \text{server\_send\_key}, [K], N)$$

The first event describes the execution of a transition called "initiator_accept_key" by honest principal A that involves a key K and some other principal B who may or may not be honest. The second event records a server's application of rule "server_send_key" relative to honest principals A and B, and key K.

Any principal can perform a "send" or a "receive" event, but only the honest principals are entitled to do an "accept" event. As we will see below, events are the building blocks of the NPATRL language.

A Requirement Language for the NRL Protocol Analyzer

The NRL Protocol Analyzer model described above has successfully been used to verify a number of protocols, sometimes uncovering previously unknown flaws [Mea94,Mea96]. This is all the more laudable once we acknowledge the implicit and rudimentary manner in which requirements are entered in this system: secrecy and authentication goals are expressed as states that should not be reachable from the initial state. This unintuitive and occasionally error prone way of writing requirements would have made it very difficult to use the NRL Protocol Analyzer for large protocols.

The *NRL Protocol Analyzer Temporal Requirements Language*, better known as NPATRL (and pronounced "N Patrol"), was designed to address these shortcomings [SM96]. This formalism makes available the abstract expressiveness of a logical language to specify requirements at a high enough level to capture intuitive goals precisely, and yet it can be interpreted in the NRL Protocol Analyzer search engine.

NPATRL requirements are logical expressions whose atomic formulas are *event statements*: they include the "receive", "accept" and "send" events that can be found in the trace of an NRL Protocol Analyzer search, and the special "learn" event that indicates the acquisition of information by the adversary. The logical infrastructure of NPATRL consists of the usual connectives $\neg$, $\wedge$, $\rightarrow$, etc, and the temporal modality $\diamondminus$ which, similarly to what we saw in Section 4.3, is interpreted as "happens before" or "previously".

For example, we may have the following requirement:

> *If an honest principal A accepts a key K for communicating with another honest principal B, then a server must have previously generated and sent this key with the idea that it should be used for communications between A and B, and that both are expected to be honest.*

We can use the NRL Protocol Analyzer events given in the previous section to construct an NPATRL formula that expresses it:

$$\begin{array}{ll} & \text{accept}(\text{user}(A, \text{honest}), [\text{user}(B, H)], \text{initiator\_accept\_key}, [K], N) \\ \rightarrow & \diamondminus\ \text{send}(\text{server}, [\text{user}(A, \text{honest}), \text{user}(B, \text{honest})], \text{server\_send\_key}, [K], N) \end{array}$$

This formula is a simple expression of the above requirement. A direct encoding in terms of final states is tricky, in particular if we want to faithfully express the temporal meaning of the operator "$\diamondminus$".

Intuitively, the protocol verification process changes from what we discussed in the previous section by using NPATRL requirements where the final state appeared. More precisely, we first need to map every NPATRL event statement to an actual event in the NRL Protocol Analyzer specification of the protocol. Then, we take the negation of each NPATRL requirement as a way to characterize the states that should be unreachable if and only if that requirement is satisfied. At this point, we perform the analysis as in the previous section: if the NRL Protocol Analyzer proves that this goal is unreachable, the protocol satisfies the original requirement. Otherwise, it returns a trace corresponding to a attack on the protocol that potentially invalidates the requirement.

Abstractly, the verification process of the NRL Protocol Analyzer enhanced with the NPATRL language can be expressed by the following diagram:

$$\begin{tikzcd} & \text{Negated NPATRL requirements} \arrow[d] & \\ \text{NPA protocol specification} \arrow[r] & \boxed{\text{NPA checker}} \arrow[r] & \text{OK / Attack} \end{tikzcd}$$

NPATRL has been extensively used in the last few years to analyze protocols with various characteristics. Among these, generic requirements have been given for two-party key distribution protocols [SM93,SM94] and two-party key agreement protocols [SM96]. The most ambitious specification undertaken using NPATRL has involved the requirements of the credit card payment transaction protocol SET (Secure Electronic Transactions) [MS98]. SET proved particularly difficult to specify for several reasons. First, nowhere in its hefty documentation (indeed, about 50cm thick) [SET97] are the requirements of this protocol stated, even informally. Second, it relies on some unfamiliar constructs such as dual signatures. Finally, the objects to be authenticated are dynamic: unlike keys, what is agreed upon changes as it passes from one principal to another. This exercise revealed several ambiguities [MS98].

6.2 Strand Semantics for BAN languages

In the last section, we presented a case study that separated the syntax in which requirements are best stated (NPATRL) from the semantics in which the protocol is best specified and evaluated (NPA). In this section we explore the possibility of a similar strategy for BAN-style languages. (The content of this section is largely taken from [Syv00].)

Some BAN-like logics already have a model-theoretic semantics, for example, AT and SVO. Such a semantics can provide assurance in the reasoning embodied in a logic, via a soundness result. However, as illustrated in the last section, it can also provide another level on which to reason. These points were alluded to in Sections 2.4 and 3.5. And, providing an independently motivated model-theoretic semantics for BAN was a central design idea underlying the development of both AT and SVO. But, the model of computation in the semantics for each of these was adapted from general models underlying epistemic logics to reason about distributed computing. Their primary focus was not authentication or even cryptographic protocols generally. It is perhaps not surprising, therefore, that previous analysis showed AT and SVO computational models not easily compatible with those of NPA [Mea94,Syv98].

Perhaps what is needed is a model of computation that is more directly intended to represent authentication protocols. One such model is strand spaces [THG97,THG98b]. (See also [Gut] in this volume. Related to strands is the multiset rewriting (MSR) approach [CDL$^+$].) Besides being a model specifically directed at this problem area and having a growing base of theoretical literature, it seems to fit somewhat naturally to NPA and similar tools, e.g., Athena [Son99]. The question that naturally arises is then whether we can effectively repeat the above NPATRL idea using something like BAN for the requirements language and strands as the model. In other words, could we have a process as expressed in the following diagram?

$$\begin{tikzcd}
 & \text{BAN-style authentication goals} \arrow[d] & \\
\text{Strand protocol specification} \arrow[r] & \boxed{\text{Strand machine}} \arrow[r] & \text{OK / Attack}
\end{tikzcd}$$

An affirmative answer would require a strand semantics for a BAN-style language. We will present a proposal for one below. We shall first provide a brief overview of the relevant strand space concepts.

Overview of Strands

A strand is basically a local history of sent and received messages in a protocol run. A strand space is a collection of strands, and a bundle is a graph that reflects a causally meaningful way that a set of strands might be connected.

The messages sent between principals are taken from an algebra A of terms. We will say more about the algebra shortly. Terms can be signed, e.g., $+t$ or $-t$, to indicate sending and receiving of messages respectively. We will give definitions for all the relevant concepts below. First, here is a picture of a bundle for Protocol 7, the (abridged) Needham-Schroeder Public-Key Protocol.

$$\begin{tikzcd}
+\{n_A, A\}_{k_B} \arrow[r] \arrow[d, Rightarrow] & -\{n_A, A\}_{k_B} \arrow[d, Rightarrow] \\
-\{n_A, n_B\}_{k_A} \arrow[r] \arrow[d, Rightarrow] & +\{n_A, n_B\}_{k_A} \\
+\{n_B\}_{k_B} &
\end{tikzcd}$$

The vertical sequences of double arrows are the strands, the local traces of messages sent to and from a given principal (in a given run). The horizontal (single) arrows link one strand to another by connecting the transmission and the reception of the same message. We now give more precise definitions, all of which are taken from [THG99b].

Let Σ be a set of strands and $(\pm\mathsf{A})^*$ be the set of all finite sequences of signed terms. A *strand space* over A is a set Σ together with a trace mapping $tr : \Sigma \to (\pm\mathsf{A})^*$.

Fix a strand space Σ

1. A *node* is a pair $\langle s, i\rangle$, with $s \in \Sigma$ and i an integer satisying $1 \le i \le \mathrm{length}(\mathrm{tr}(s))$. The set of nodes is denoted by $\mathcal{N}$. We will say the node $\langle s, i\rangle$ belongs to the strand s. Clearly, every node belongs to a unique strand.
2. If $n = \langle s, i\rangle \in \mathcal{N}$ then $\mathrm{index}(n) = i$ and $\mathrm{strand}(n) = s$. Define $\mathrm{term}(n)$ to be $(\mathrm{tr}(s))_i$, i.e. the ith signed term in the trace of s. Similarly, $\mathrm{uns\_term}(n)$ is $((\mathrm{tr}(s))_i)_2$, i.e. the unsigned part of the ith signed term in the trace of s.
3. There is an edge $n_1 \to n_2$ if and only if $\mathrm{term}(n_1) = +a$ and $\mathrm{term}(n_2) = -a$ for some $a \in \mathsf{A}$. Intuitively, the edge means that node n_1 sends the message a, which is received by n_2, recording a potential causal link between those strands.
4. When $n_1 = \langle s, i\rangle$ and $n_2 = \langle s, i+1\rangle$ are members of $\mathcal{N}$, there is an edge $n_1 \Rightarrow n_2$. Intuitively, the edge expresses that n_1 is an immediate causal predecessor of n_2 on the strand s. We write $n' \Rightarrow^+ n$ to mean that n' precedes n (not necessarily immediately) on the same strand.
5. $\mathcal{N}$ together with both sets of edges $n_1 \to n_2$ and $n_1 \Rightarrow n_2$ is a directed graph $\langle \mathcal{N}, (\to \cup \Rightarrow)\rangle$.

Suppose $\to_\mathcal{C} \subseteq \to$; suppose $\Rightarrow_\mathcal{C} \subseteq \Rightarrow$; and suppose $\mathcal{C} = \langle \mathcal{N}_\mathcal{C}, (\to_\mathcal{C} \cup \Rightarrow_\mathcal{C})\rangle$ is a subgraph of $\langle \mathcal{N}, (\to \cup \Rightarrow)\rangle$. $\mathcal{C}$ is a *bundle* if:

1. $\mathcal{C}$ is finite.
2. If $n_2 \in \mathcal{N}_\mathcal{C}$ and $\mathrm{term}(n_2)$ is negative, then there is a unique $n_1 \in \mathcal{N}_\mathcal{C}$ such that $n_1 \to_\mathcal{C} n_2$.
3. If $n_2 \in \mathcal{N}_\mathcal{C}$ and $n_1 \Rightarrow n_2$ then $n_1 \Rightarrow_\mathcal{C} n_2$.
4. $\mathcal{C}$ is acyclic.

If $\mathcal{S}$ is a set of edges, i.e. $\mathcal{S} \subseteq (\to \cup \Rightarrow)$, then $\prec_\mathcal{S}$ is the transitive closure of $\mathcal{S}$, and $\preceq_\mathcal{S}$ is the reflexive and transitive closure of $\mathcal{S}$. The relations $\prec_\mathcal{S}$ and $\preceq_\mathcal{S}$ are each subsets of $\mathcal{N}_\mathcal{S} \times \mathcal{N}_\mathcal{S}$, where $\mathcal{N}_\mathcal{S}$ is the set of nodes incident with any edge in $\mathcal{S}$.

These are all of the definitions that we need to set out a possible worlds model and semantics for sending, receiving, and knowledge. We will provide below more details about the term algebra that will allow us to express, e.g., that a principal who receives a ciphertext (encrypted message) and has the decryption key has also got the unencrypted message.

Possible Worlds from Strand Spaces

We now describe a possible world semantics of epistemic logics for distributed computing in general and for security protocols in particular, for example, as presented in [AT91,SvO94,SvO96].

In a traditional system model and knowledge semantics for distributed computing, computation is performed by a finite set of principals, $P_1, \ldots, P_n$, who send messages to one another. In addition there is a principal P_e representing the environment. This allows modeling of any penetrator actions as well as reflecting messages in transit.

Each principal P_i has a local state s_i. A global state is thus an $(n+1)$-tuple of local states.

A run is a sequence of global states indexed by integers to represent time. The first state of a given run r is assigned a time $t_r \leq 0$. The initial state of the current authentication is at $t = 0$. The global state at time t in run r determines a possible world (sometimes also called nodes or points). We assume that global states are unique wrt runs and times. Thus, they can be referred to by, e.g., '$\langle r, t\rangle$'. At any given global state, various things will be true, e.g., that principal Q has previously sent the message $\{X\}_k$. What a principal P then knows (believes) at a given point $\langle r, t\rangle$ is precisely that which is true at all possible worlds with the same local state $r_P(t)$ for P as $\langle r, t\rangle$. This is typically captured by means of an accessibility relation on global states $\rightsquigarrow_P$ for a principal P. When the relation is an equivalence, it is also called an indistinguishability relation $\sim_P$ for a principal P. This allows for a simple intuitive definition, without even having to describe in any way properties of local states, viz:

– $\langle r, t\rangle \sim_P \langle r', t'\rangle$ iff P is in the same local state at both points, i.e., $r_P(t) = r'_P(t')$.

Given an indistinguishability relation, we can then go on to define principal P's knowledge in terms of the worlds that are P-indistinguishable.

– $\langle r, t\rangle \models P \; \textit{knows} \; \varphi$ iff $\langle r', t'\rangle \models \varphi$ for all $\langle r', t'\rangle$ such that $\langle r, t\rangle \sim_P \langle r', t'\rangle$

The above system model and characterization of knowledge (belief) is essentially what is found in [AT91,SvO94,SvO96]. It is largely based on similar models and characterizations of knowledge in distributed computing; see for example [FHMV95]. Note that the relation just given is an equivalence relation, as is the strand-based relation to be given presently. For this reason, and to be consistent with earlier literature such as [FHMV95], we refer to the associated modality as knowledge rather than belief, but no great significance should be attached to this choice, as we saw in Section 3.3. We now turn specifically to strand spaces as a basis for knowledge semantics.

Strand Semantics for Knowledge

In the conclusion of [THG97] it was suggested that, "[what] a protocol participant knows, in virtue of his experience in executing a protocol, is that he has performed the actions lying on some strand s. Thus, the real world must include some bundle $\mathcal{C}$ such that s is contained in $\mathcal{C}$. The beliefs that the participant may justifiably hold are those that are true in every bundle $\mathcal{C}$ containing s." [THG97]

Thus, a possible world on this approach is simply a bundle. This is a reasonable approach for reasoning about some protocol features. However, we found it also worthwhile to include in the definition of possible worlds the nodes within bundles. We did this in order to capture temporal aspects of the above authentication logics, specifically freshness. This will also facilitate the addition of richer temporal formulae to the logic, as in [Syv93a].

Neither strand spaces nor bundles have a notion of global time. Thus we cannot have an indistinguishability relation that corresponds directly to the above. However, $\langle \mathcal{C}, s, i\rangle$ picks a unique point $\langle s, i\rangle$ in bundle $\mathcal{C}$ and partitions $\mathcal{N}_{\mathcal{C}}$ into $\{\langle t, j\rangle : \langle t, j\rangle \preceq_{\mathcal{C}} \langle s, i\rangle\}$ and $\{\langle t, j\rangle : \langle t, j\rangle \not\preceq_{\mathcal{C}} \langle s, i\rangle\}$. This partition allows us to define an accessibility relation on nodes in bundles based on local time.

1. Given a strand s, let $\mathrm{princ}(s)$ refer to the principal whose strand s is.
2. Given a node $\langle s, i\rangle$ and a strand t in a bundle $\mathcal{C}$, let the *restriction of t to $\langle s, i\rangle$ in $\mathcal{C}$* be $\mathrm{tr}(t) \restriction \langle s, i\rangle = \langle \mathrm{tr}(t)_1, \ldots, \mathrm{tr}(t)_j\rangle$, where $\langle t, j\rangle$ is the greatest node on t s.t. $\langle t, j\rangle \preceq_{\mathcal{C}} \langle s, i\rangle$.

With this notation in place we can now define an indistinguishability relation.

Assume bundles $\mathcal{C}, \mathcal{C}'$, and strands s, s', and indices i, i' such that $\langle s, i\rangle \in \mathcal{N}_{\mathcal{C}}$ and, $\langle s', i'\rangle \in \mathcal{N}_{\mathcal{C}'}$. A natural definition, analogous to the runs-and-times definition of the traditional literature would be to have $\langle \mathcal{C}, s, i\rangle \sim_P \langle \mathcal{C}', s', i'\rangle$ (i.e., $\langle \mathcal{C}, s, i\rangle$ is *P-indistinguishable* from $\langle \mathcal{C}', s', i'\rangle$) just in case P's history in $\mathcal{C}$ up to $\langle s, i\rangle$ matches P's history in $\mathcal{C}'$ up to $\langle s', i'\rangle$. This is exactly right. However, just as there is no global time in a bundle, there may also be multiple strands associated with one principal. The resulting definition is thus:

$\langle \mathcal{C}, s, i\rangle$ is *P-indistinguishable* from $\langle \mathcal{C}', s', i'\rangle$ (written as $\langle \mathcal{C}, s, i\rangle \sim_P \langle \mathcal{C}', s', i'\rangle$) iff

1. for any t in $\mathcal{C}$ s.t. $\mathrm{princ}(t) = P$ there exists t' in $\mathcal{C}'$ s.t. $\mathrm{tr}(t) \restriction \langle s, i\rangle = \mathrm{tr}(t') \restriction (s', i')$ and $\mathrm{princ}(t') = P$, and
2. the number of strands satisfying clause 1 is the same in $\mathcal{C}$ and $\mathcal{C}'$.

Truth Conditions for BAN-Style Formulae

The purpose of this section, is to present truth conditions for basic formulae of a BAN-style language. The basic notions we cover are freshness, key goodness, said and received (got) messages, and jurisdiction.

Given our definition of $\sim_P$ above we can now present truth conditions for knowledge in this semantics. Let φ be some formula in our language. We will define $\models$ inductively; however the presentation is organized pedagogically rather than to respect the inductive construction. We assume the usual truth conditions for logical connectives; although we will not discuss compound formulae here.

$$\langle \mathcal{C}, s, i\rangle \models P \textit{ knows } \varphi$$

iff $\langle \mathcal{C}', s', i'\rangle \models \varphi$ at all $\langle \mathcal{C}', s', i'\rangle$ s.t. $\langle \mathcal{C}, s, i\rangle \sim_P \langle \mathcal{C}', s', i'\rangle$

This definition gives a strand semantics for knowledge in a distributed environment. However, we have not yet described what specific types of things φ might express. Giving truth conditions for the various possibilities is the focus of the remainder of this section.

We can give semantics for formulae expressing the sending and receiving of messages without giving any more details about the model. Let M be an arbitrary message from our term algebra A. Then,

$$\langle \mathcal{C}, s, i\rangle \models P \textit{ sent } M$$

iff there is a node $\langle t, j\rangle$ in $\mathcal{C}$ s.t. (i) $\text{princ}(t) = P$, (ii) $\langle t, j\rangle \preceq \langle s, i\rangle$, and (iii) $\text{term}(\langle t, j\rangle) = +M$. Moreover,

$$\langle \mathcal{C}, s, i\rangle \models P \textit{ received } M$$

iff there is a node $\langle t, j\rangle$ in $\mathcal{C}$ s.t. (i) $\text{princ}(t) = P$, (ii) $\langle t, j\rangle \preceq \langle s, i\rangle$, and (iii) $\text{term}(\langle t, j\rangle) = -M$.

To give the truth conditions for other formulae, we must first spell out some of the structure of the term algebra and define a notion of submessage. The following definitions are taken from [THG99b] and can also be found in the preceding strand space papers.

Assume the following:

– A set $\mathbf{T} \subseteq \mathsf{A}$ of texts (representing the atomic messages).
– A set $\mathbf{K} \subseteq \mathsf{A}$ of cryptographic keys disjoint from $\mathbf{T}$, equipped with a unary operator $\mathsf{inv} : \mathbf{K} \to \mathbf{K}$.
  inv is injective; i.e., that it maps each member of a key pair for an asymmetric cryptosystem to the other; and that it maps a symmetric key to itself.
– Two binary operators

$$\mathsf{encr} : \mathbf{K} \times \mathsf{A} \to \mathsf{A}$$
$$\mathsf{join} : \mathsf{A} \times \mathsf{A} \to \mathsf{A}$$

We will follow notational conventions, some of which have already been mentioned, and write $\mathsf{inv}(k)$ as k^{-1}, $\mathsf{encr}(k, M)$ as $\{M\}_k$, and $\mathsf{join}(a, b)$ as $(a\,b)$. If K is a set of keys, K^{-1} denotes the set of inverses of elements of K.

The next assumption we make is that A is the algebra freely generated from $\mathbf{T}$ and $\mathbf{K}$ by the two operators encr and join. As noted in [THG99b], this assumption has been commonly made in this area of research going back to [DY83]. As in [THG99b] it is probably stronger than what we ultimately need but is pedagogically convenient. Amongst other things, it implies that encryptions and concatenations are unique and always distinct from each other and from **T** and **K**.

Central to the semantics of *said* formulae is the concept of an ideal. Interestingly, in the strand space papers, it was introduced to formulate general facts about the penetrator's capabilities; while, for this discussion, we will say virtually nothing about the nature of the penetrator.

If $K \subseteq \mathbf{K}$, a K-ideal of A is a subset I of A such that for all $h \in I$, $g \in \mathsf{A}$ and $k \in K$

1. $h\,g, g\,h \in I$.
2. $\{h\}_k \in I$.

The smallest K-ideal containing h is denoted $I_K[h]$.

The notion of ideal can be used to define a subterm relation $\sqsubset$ as follows [THG98a].

Let $K \subseteq \mathbf{K}$. $s \in \mathsf{A}$ is a *K-subterm of* $t \in \mathsf{A}$, $(s \sqsubset_K t)$ iff $t \in I_K[s]$.

If $K = \mathbf{K}$ in this definition, then we say simply that s *is a subterm of* t, and write $s \sqsubset t$.

We now give truth conditions for *said* formulae

$$\langle \mathcal{C}, s, i\rangle \models P \; said \; M$$

iff there is a message M' s.t. $\langle \mathcal{C}, s, i\rangle \models P \; sent \; M'$ and $M \sqsubset_K M'$ where K is the set of keys possessed by P at $\langle s, i\rangle$.

Notice that P is held accountable, e.g., for saying M at n, if he sends $\{M\}_k$ at $n' \preceq n$ and he has k at n, even if k was not in his key set until some n'' s.t. $n' \prec n'' \preceq n$.

A definition that does not occur in any of the strand space papers is that of a filter. In many contexts, filters are the duals of ideals. In our case, they are useful for giving semantics to *got* formulae, those that express the understood messages contained in received messages. (Millen and Rueß introduce the same idea in [MR00] to reason about secrecy invariants. They call it a "coideal".)

If $K \subseteq \mathbf{K}$, a K-filter of A is a subset F of A such that for all $h, g \in \mathsf{A}$ and $k \in K$

1. $h\,g \in F$ implies $h \in F$ and $g \in F$
2. $\{h\}_k \in F$ implies $h \in F$ for $k^{-1} \in K$

The smallest K-filter containing h is denoted $F_K[h]$.

In general, the relation between filters and ideals is not so simple because, in public-key cryptography, one may have k and not have k^{-1}, or vice versa. However, in this section we are limiting discussion to the symmetric key case, $k = k^{-1}$—for which there is a simple relation. (This relation also holds when both cognates of a public/private key pair are known.) It is easy to show that

Claim 4 *For all sets of keys K' of the form $K \cup K^{-1}$*

$$g \in F_{K'}[h] \quad \textit{iff} \quad h \in I_{K'}[g].$$

Thus, for key sets K' of this form, by definition 6.2, $s \sqsubset_{K'} t$ iff $s \in F_{K'}[t]$. We can now give the truth conditions for *got* formulae. (We present them for the general case.)

$$\langle \mathcal{C}, s, i \rangle \models P \textit{ got } M$$

iff there is a message M' s.t. $\langle \mathcal{C}, s, i \rangle \models P \textit{ received } M'$ and $M \in F_K[M']$ where K is the set of keys possessed by P at $\langle s, i \rangle$.

We can use the truth conditions for *said* and *got* formulae to further give the truth conditions for key goodness.

$$\langle \mathcal{C}, s, i \rangle \models P \overset{k}{\longleftrightarrow} Q$$

iff, for all $\langle s', i' \rangle \in \mathcal{N}_{\mathcal{C}}$, $\langle \mathcal{C}, s', i' \rangle \models R \textit{ said } \{M \textit{ from } Q\}_k$ implies either
$\langle \mathcal{C}, s', i' \rangle \models R \textit{ received } \{M \textit{ from } Q\}_k$, or $R = Q$ and $\langle \mathcal{C}, s', i' \rangle \models R \textit{ said } M$.
Moreover, if $\langle \mathcal{C}, s', i' \rangle \models R \textit{ said } \{M\}_k$
(instead of the stronger $\langle \mathcal{C}, s', i' \rangle \models R \textit{ said } \{M \textit{ from } Q\}_k$), then $R \in \{P, Q\}$
(instead of the stronger $R = P$).

Note that these are the truth conditions from [SvO96] with $\langle \mathcal{C}, s, i \rangle$ replacing $\langle r, t \rangle$ and $\langle \mathcal{C}, s', i' \rangle$ replacing $\langle r, t' \rangle$ throughout. This was itself based on the truth conditions for goodness given in [AT91].

Once we have a mechanism to express the beginning of the current epoch, we will be able to similarly dispatch the freshness and jurisdiction formulae. In order to do that, we must again confront the absence of a global concept of time. In the system models for possible world semantics of BAN-like logics, it was trivial to stipulate a global time t_0 and then define something as fresh if it was not said (by anyone) prior to t_0. We instead define a concept now as follows.

For any bundle $\mathcal{C}$, $\mathsf{now}_{\mathcal{C}} \subseteq \mathcal{N}_{\mathcal{C}}$, is a nonempty set of incomparable nodes (i.e., a nonempty set of nodes s.t. $n, n' \in \mathsf{now}_{\mathcal{C}}$ implies $n \not\preceq n'$ and $n' \not\preceq n$). For $n \in \mathcal{N}_{\mathcal{C}}$, we may write '$\mathsf{now}_{\mathcal{C}} \preceq n$' just in case there exists $n' \in \mathsf{now}_{\mathcal{C}}$ s.t. $n' \preceq n$. When it is clear from context which bundle is relevant, we will write simply 'now'.

Thus,

$$\langle \mathcal{C}, s, i \rangle \models \textit{fresh}(M)$$

iff for all principals P, $\langle \mathcal{C}, s', i' \rangle \models P \textit{ said } M$ implies $\mathsf{now} \preceq \langle s', i' \rangle$.

The truth conditions for jurisdiction assume truth conditions for *says* formulae, which the definition of $\mathsf{now}_{\mathcal{C}}$ allows us to formulate.

$$\langle \mathcal{C}, s, i \rangle \models P \textit{ says } M$$

iff there is a message M' and a node $\langle t, j \rangle$ in $\mathcal{C}$ s.t. (i) $\text{princ}(t) = P$, (ii) $\mathsf{now} \preceq \langle t, j \rangle \preceq \langle s, i \rangle$, (iii) $\text{term}(\langle t, j \rangle) = +M'$, and (iv) $M \sqsubset_K M'$ where K is the key set possessed by P at $\langle s, i \rangle$.

If φ is a formula.

$$\langle \mathcal{C}, s, i \rangle \models P \textit{ controls } \varphi$$

iff $\langle \mathcal{C}, s, i \rangle \models P \, \textit{says} \, \varphi$ implies $\langle \mathcal{C}, s', i' \rangle \models \varphi$ for any $\langle s', i' \rangle$ s.t. $\mathsf{now} \preceq \langle s', i' \rangle$.

These conditions are similar to those in [AT91] and [SvO94,SvO96], *mutatis mutandis*. Notice that goodness is a condition that is constant across all points in the same bundle. And, jurisdiction and freshness are constant across all points in the present epoch. Notice also that jurisdiction is restricted to those messages that are formulae, rather than messages in general.

This completes our presentation of truth conditions. Strand based truth conditions for public keys, Diffie-Hellman, and other aspects of SVO have yet to be developed. What we have done is to provide a means by which BAN-style requirements can be mapped to strand-style protocol specifications. Something like this is necessary for the protocol analysis approach characterized by the diagram at the beginning of this section. For the "strand machine" to process its inputs there must be some means for it to combine them. The mapping provides such a means. To completely develop the semantic approach using BAN-style requirements for a strand-style model, the strand machine itself must be built. We conclude with a description of some of other areas where there is still much work to be done.

7 The Future

In [Mea00b], Meadows sets out a number of open areas in the application of formal methods to cryptographic protocols. The primary focus is beyond simple two-party authentication protocols, and that paper is a good place to get an idea of where much of the cutting edge research is or soon will be. We finish up with a discussion of these open areas, but with a slant towards the kinds of formalisms and ideas that have been discussed above. We also try to mention some of the recent work which has not been alluded to elsewhere above. Indeed, such a large amount of work has been done in formal analysis of authentication and other security protocols that, despite the number of references cited herein, far more work has gone unmentioned, much of it quite good.

Appropriately, one of the open areas is in *open-ended protocols*. The two major ways in which a protocol can be open-ended is in what data is sent and in who is sending or receiving. We address these in order.

A protocol may be open-ended in virtue of the data sent. For example, the Internet Key Exchange Protocol (IKE) that is part of IPSEC [DH99], includes an agreement on a Security Association (SA). The SA includes such things as a choice of algorithms and other parameters. But, there is no defined (upper) limit on what can be included in an SA. This sort of open-endedness has not been formally analyzed as far as we know. Another aspect of IKE is that the SA has a more elaborate, indeed open-ended, data structure than a simple cryptographic key. In classic authentication protocols, the data about which we prove authentication and secrecy properties is simply a key. In Diffie-Hellman exchange, there may be parts contributed by the principals that make up the key, but Diffie-Hellman based protocols have been analyzed using VO and SVO [vO93,SvO96].

Protocols can also be open-ended in the participants involved. An obvious example is in various kinds of group protocols. These can be for both group authentication and group confidentiality properties. One example of such a group protocol is a group signature protocol, in which a signature can identify only that the signer was from a group unless an additional protocol is run (typically with a trusted authority) to reveal the individual responsible for the given signature. As introduced in [CvH91], these were perhaps only open-ended in principle since there was no efficient means to add members to an existing group. The first significant advance on that problem was made in [CS97], and others have since followed. There has not been any formal methods work that we know of directly on this area. More positive results have been seen in the area of group Diffie-Hellman [AST98]. These are essentially Diffie-Hellman type establishments for open-ended groups. Meadows was able to analyze these protocols after expanding NPA [Mea00a]. More recently, Meadows has presented evaluations of secure multicast to the IETF and the IRTF Secure Multicast Group. We have begun specification and examination of secure multicast protocols using NPATRL. Other formal methods work involving groups of arbitrary size can be found in [Pau97,BS97]. Both of these papers make use of theorem proving, Isabelle and PVS respectively to examine the same protocol.

Another important open are is *denial of service*. Meadows has devised a framework [Mea01] for reasoning about denial of service in cryptographic protocols, although not a formal method per se. The problem with authentication is that it is not only a protection against but a great source of denial-of-service attacks. If only authenticated principals are allowed to perform any actions, then unauthenticated principals cannot deny service. But, verifying authentication typically involves computationally intensive cryptographic operations. Thus, initiating many authentic connections can be an even more effective denial-of-service attack than simply initiating many connections. Meadows builds on the fail-stop concept set out in Section 5.3. The idea is to have the amount of work expended to defend a protocol against denial of service increase as the protocol progresses. The protocol is analyzed to show that it is fail-stop against an attacker whose capabilities are within a specified constraint. Note that this is a diversion from the Dolev-Yao intruder model that we have assumed throughout, up to this point. Obviously a Dolev-Yao intruder can arbitrarily deny service. Much of the open work involves backing off from such an unrealistically strong attacker to consider properties that can be established in the face of a different attacker.

Electronic commerce, in particular non-repudiation and fair exchange, is an area that has seen an explosion of protocols and also some formal work in the last several years. In fair exchange, there is no adversary per se. Rather, the idea is to make sure that each party gets his goods, signed contract, etc. just in case the other does as well. In non-repudiation, the goal is to have evidence that a principal cannot repudiate. This can be evidence of messages sent (evidence of origin) or messages received (evidence of receipt). Obviously fair exchange and non-repudiation are closely related. The first attempt to reason about this area formally was by Kailar using a BAN-like logic [Kai95,Kai96]. The central logical construct is CanProve as in "A CanProve B says X". Zhou and Gollman also used SVO to reason about non-repudiation properties [ZG98]. We have already mentioned Brackin's verification of the Cybercash main sequence protocol using BGNY [Bra97]. A more recent approach to non-repudiation, using temporal logic with a game semantics can be found in [KR00].

The SET protocol is a good illustrator of several of the complexities we have introduced in this section. Like IKE, it is not a single protocol but a collection of subprotocols. As mentioned in Section 6.1, the protocol is very large and complex with many options, yet its specification lacks even an informal statement of requirements. It has a more elaborate structure than just a key on which principals must agree: there is a transaction on which the customer, merchant, and bank must agree, but parts of the transaction are hidden from some of the principals and parts are added to it as the protocol progresses. And, the reason that parts of the transaction are hidden is because the principals are mutually mistrusting and attempting some sort of non-repudiable fair exchange. Nonetheless, NPATRL was adapted to express requirements for payments in SET and related protocols by adding abstract structures for which some of the components are not revealed [MS98]. Also, the cardholder registration subprotocol has been verified using Isabelle and HOL [BMPT00]. Recall that in SVO and the AUTLOG based logic of [WK96] one can reason about principals' beliefs concerning messages in which not all the parts are recognizable. This would seem naturally generalizable to SET. In [KN98], Kessler and Neuman devised a logic for reasoning about payment in SET that combine elements from these logics, from Kailar's logic of accountability, and from the Stubblebine-Wright logic of recent security [SW96].

These large protocol suites raise still another open issue: *protocol composability.* The fail-stop protocols of Section 5.3 constitute one answer to this problem. But are there less onerous design restrictions that can be imposed (similar constraints on composition are given in [HT96])? It might seem that protocol composability is completely guaranteed by having only EFS protocols. However, even when the protocols are all EFS, the application environment generally will not be. Thus, there are still oracles available for active attacks.

Suppose that principals are willing to use keys obtained through a key-distribution protocol before the protocol completes. This is sometimes called "eager" use of keys in the literature. Only if the authentication protocol does not complete within some reasonable timeout is there an alarm or noting of anomaly in the logs. This eagerness might be all the more reasonable if the protocol distributing the keys is EFS. In this case, there would seem to be no possibility of mistake about who the session key is for, who the relevant principals are, or the roles they each play (i.e., initiator or responder). But, allowing eager use of keys in an application that authenticates a random challenge by encryption using the session key could be used to attack the protocol. (This could be a variant of the sensor example of Protocol 6.)

Specifically, suppose Alice begins NSSK (Protocol 1) for a session with Bob, the attacker prevents the third message from arriving. Then, for the application challenge-response he produces:

Application Message 1	$E_B \rightarrow A : n_B$
Application Message 2	$A \rightarrow E_B : \{n_B\}_{K_{ab}}$

The attacker uses the response from Alice for the fourth message in NSSK, and intercepts the final message from Alice to Bob. Alice will now be spoofed into thinking she has completed a handshake with Bob when Bob was never present.

This attack is even possible if NSSK is strengthened to be made EFS. The point is to show that the applications that use keys established in an authentication protocol must also be considered. This aspect of protocol composability has received only a little attention. A version of this attack and related issues are discussed in [CMS01]. Besides general composable protocol design, there has also been a little work done into showing that particular protocols are composable [Mea99a,THG99a].

Another type of composability is between protocols and the cryptographic algorithms they employ. Protocol analysis as we have described it herein has treated cryptography as a black box, but some protocols and algorithms are secure if used in one combination while they are insecure in different combinations. Formal work going beyond black box treatments of cryptography in protocol analysis is just beginning [AR00,Can00,Jür00].

We mentioned the inappropriateness of Dolev-Yao adversaries for modeling denial-of-service attacks. They are also clearly inadequate for exchange protocols involving mutually mistrusting parties. Another area in which a Dolev-Yao adversary is simply too strong is anonymity. *Anonymity* services that have either been designed to be practical for most applications or that have actually been fielded are simply broken against a Dolev-Yao adversary [Oni,Ano,Cro,Fre]. One reason is that anonymity for all of these involves passing messages through an intermediate point so as to obscure identity of an originator from anyone observing a transmission. Some involve hopping through several points and some change the appearance of messages at each point so that parts of the transmission cannot be compared and seen to *be* parts of the same transmission. No matter how many of these precautions are taken, in a system where all messages pass through the intruder, the intruder will know exactly who is talking to whom (and possibly what is being said unless confidentiality is also protected). There are communication mechanisms that are secure against a Dolev-Yao intruder, e.g., dining cryptographer (DC) nets [Cha88]. However, nothing that is practical for widely used email, Web browsing, remote login, etc. is secure against a Dolev-Yao intruder. In [SS99], an epistemic model and logic was introduced for reasoning about group principals. This built on ideas in [FHMV95]. Recall from Section 6.2 that the usual model of computation associated with these logics has a single principal to represent the environment/penetrator. This is in perfect keeping with the Dolev-Yao model. However, in [SS99], all communication principals, including the environment must be specified. And, there is in fact no single environment. Rather, there are many environment principals that have various capabilities and properties and that can be assembled in a variety of ways, i.e., into various sorts of group principals. One can then reason about various properties associated with a group of principals (the 'good guys') that another group of principals (the intruder) can actively or passively determine. For example, a particular distributed intruder may be able to determine that some (atomic) subprincipal of a group principal of cardinality n was the source of a message, but cannot narrow the cardinality lower than n. Work is underway to combine this approach, which has an intuitive yet formal expressiveness, with a CSP based approach [SS96]. The intent is to use the CSP as a semantics, much as the strand semantics for BAN described in Section 6.2. The language in [SS99] includes threshold-group principals and other primitives that should make it applicable to other areas besides anonymity.

Childhood's End

Specification and analysis of basic authentication protocols has been the focus of much of the above discussion—and much of the work in the last dozen years of formal methods in application to cryptographic protocols. The main concepts have been extensively explored and both intuitive and fully automated techniques have been developed, techniques that now do a thorough job and require no great sophistication. It has been several years since merely documenting a new attack on such protocols or devising a new formal method for reasoning about them was sufficient for publication in even small workshops. This is a positive sign. More complex protocols and protocols to accomplish more ambitious and subtle goals continue to come along. Formal methods are increasingly employed in the specification and analysis of protocols that are more than academic exercises: commercial products, complex protocol suites, international standards, etc. And, they have begun to have an impact in the real-world protocols that are being deployed. At the same time there has been a resurgence in theoretical models of both the new and the classic concepts, and these have in turn influenced the development and refinement of formal methods for protocol analysis and even design. It's an exciting time to be in the field.

References

[ABKL90] M. Abadi, M. Burrows, C. Kaufman, and B. Lampson. Authentication and delegation with smart-cards. Research Report 67, Digital Systems Research Center, October 1990. Revised July, 1992.

[AG97] Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: the spi calculus. In *Proceedings of the Fourth ACM Conference on Computer and Communications Security*, pages 36–47. ACM Press, April 1997.

[AG99] Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: the spi calculus. *Information and Computation*, 143:1–70, 1999. An extended version of this paper appears as Research Report 149, Digital Equipment Corporation Systems Research Center, January 1998. An early presentation appears in [AG97].

[AN94] Martín Abadi and Roger Needham. Prudent engineering practices for cryptographic protocols. In *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, pages 122–136. IEEE CS Press, May 1994.

[AN95] Ross Anderson and Roger Needham. Robustness principles for public key protocols. In D. Coppersmith, editor, *Advances in Cryptology — CRYPTO '95*, pages 236–247. Springer-Verlag, LNCS 963, August 1995.

[AN96] Martín Abadi and Roger Needham. Prudent engineering practices for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, January 1996. A preliminary version appeared as [AN94].

[Ano] The anonymizer. http://www.anonymizer.com/.

[AR00] Martín Abadi and Phillip Rogaway. Reconciling two views of cryptographic protocols (the compuational soundness of formal encryption. In *IFIP International Conference on Theoretical Computer Science (IFIP TCS2000).* Springer-Verlag, LNCS, 2000.

[AST98] Giuseppe Ateniese, Michael Steiner, and Gene Tsudik. Authenticated group key agreement and friends. In 5^{th} *ACM Conference on Computer and Communications Security (CCS'98)*, pages 17–26. ACM Press, November 1998.

[AT91] Martín Abadi and Mark R. Tuttle. A semantics for a logic of authentication. In *Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing*, pages 201–216. ACM Press, August 1991.

[BAN88] Michael Burrows, Martín Abadi, and Roger Needham. Authentication: A practical study of belief in action. In M. Vardi, editor, *Proceedings of the Second Conference on Theoretical Aspects of Reasoning About Knowledge (Tark)*, pages 325–342. Morgan Kaufmann, March 1988. Also presented at The Computer Security Foundations Workshop, Franconia, NH, June 1988.

[BAN89a] Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication. Research Report 39, Digital Systems Research Center, February 1989. Revised Feb. 22, 1990.

[BAN89b] Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication. *Operating Systems Review*, 23(5):1–13, December 1989. This issue of *OSR*: Proceedings of the Twelfth ACM Symposium on Operating Systems Principles (SOSP), Litchfield Park, Arizona, December 1989.

[BAN89c] Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication. *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 426(1871):233–271, December 1989.

[BAN90a] Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, Feb 1990.

[BAN90b] Michael Burrows, Martín Abadi, and Roger Needham. Rejoinder to nessett. *Operating Systems Review*, 24(2):39–40, April 1990.

[BAN90c] Michael Burrows, Martín Abadi, and Roger Needham. The scope of a logic of authentication. In J. Feigenbaum and M. Merritt, editors, *Distributed Computing and Cryptography*, volume 2 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 119–126. AMS and ACM, 1990. Proceedings of a DIMACS workshop, October 1989.

[Bie90] Pierre Bieber. A logic of communication in hostile environment. In *Proceedings of the Computer Security Foundations Workshop III*, pages 14–22. IEEE CS Press, June 1990.

[BM92] Steven M. Bellovin and Michael Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, pages 72–84. IEEE CS Press, May 1992.

[BM93] Steve Bellovin and Michael Merritt. Augmented encrypted key exchange: a password-based protocol secure against dictionary attacks and password file compromise. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 244–250. ACM Press, November 1993.

[BMPT00] Giampaolo Bella, Fabio Massacci, Lawrence C. Paulson, and Piero Tramontano. Formal verification of cardholder registration in SET. In F. Cuppens, Y. Deswarte, D. Gollmann, and M. Waidner, editors, *Computer Security – ESORICS 2000*, pages 159–174. Springer-Verlag, LNCS 1895, October 2000.

[Bra96] Stephen H. Brackin. A HOL extension of GNY for automatically analyzing cryptographic protocols. In *9th IEEE Computer Security Foundations Workshop*, pages 62–76. IEEE CS Press, June 1996.

[Bra97] Stephen H. Brackin. Automatic formal analyses of two large commercial protocols. In *DIMACS Workshop on Design and Formal Verification of Security Protocols*, September 1997. Available at http://dimacs.rutgers.edu/Workshops/Security/program2/brackin.html.

[Bra98] Stephen H. Brackin. Evaluating and improving protocol analysis by automatic proof. In *11th IEEE Computer Security Foundations Workshop*, pages 138–152. IEEE CS Press, 1998.

[Bra00] Stephen H. Brackin. Automatically detecting most vulnerabilities in cryptographic protocols. In *DISCEX 2000: Proceedings of the DARPA Information Survivability Conference and Exposition*, volume I, pages 222–236. IEEE CS Press, January 2000.

[BS97] Jeremy Bryans and Steve Schneider. CSP, PVS and a recursive authentication protocol. In *DIMACS Workshop on Design and Formal Verification of Security Protocols*, September 1997. Available at http://dimacs.rutgers.edu/Workshops/Security/program2/program.html.

[BSW98] Levente Buttyán, Sebastian Staamann, and Uwe Wilhelm. A simple logic for authentication protocol design. In *11th IEEE Computer Security Foundations Workshop*, pages 153–162. IEEE CS Press, June 1998.

[Can00] Ran Canetti. A unified framework for analyzing security of protocols. Cryptology ePrint Archive, Report 2000/067, 2000. http://eprint.iacr.org/.

[Car93] Ulf Carlsen. Using Logics to Detect Implementation-Dependent Flaws. In *Proceedings of the Ninth Annual Computer Security Applications Conference*, pages 64–73. IEEE Computer Society Press, December 1993.

[Car94] Ulf Carlsen. Generating formal cryptographic protocol specifications. In *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, pages 137–146. IEEE CS Press, May 1994.

[CDL$^+$] Iliano Cervesato, Nancy Durgin, Patrick Lincoln, John Mitchell, and Andre Scedrov. A comparison between strand spaces and transition systems for the specification of security protocols. To appear as a technical report, Department of Computer Science, Stanford University.

[Cha83] David Chaum. Blind signatures for untraceable payments. In D. Chaum, R.L. Rivest, and A.T. Sherman, editors, *Advances in Cryptology – Proceedings of Crypto 82*, pages 199–203, 1983.

[Cha88] David Chaum. The dining cryptographers problem: Unconditional sender and receiver untraceability. *Journal of Cryptology*, 1(1):65–75, 1988.

[Che80] Brian F. Chellas. *Modal Logic: An Introduction*. Cambridge University Press, 1980.

[CJ97] John Clark and Jeremy Jacob. A survey of authentication protocol literature: Version 1.0, nov 1997. Available at `www-users.cs.york.ac.uk/~jac/` under the link “Security Protocols Review”.

[CJ00] John Clark and Jeremy Jacob. Searching for a solution: Engineering tradeoffs and the evolution of provably secure protocols. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 82–95. IEEE CS Press, May 2000.

[CMS01] Ran Canetti, Catherine Meadows, and Paul Syverson. Environmental requirements and authentication protocols. In *Proceedings of the Symposium on Requirements Engineering for Information Security (SREIS)*, March 2001. A version of this paper has been invited for a special issue of *Requirements Engineering*.

[Cro] Crowds. http://www.research.att.com/projects/crowds/.

[CS97] Jan L. Camenisch and Markus A. Stadler. Efficient group signature schemes for large groups. In B. Kaliski, editor, *Advances in Cryptology – CRYPTO '97*, pages 410–424. Springer-Verlag, LNCS 1294, 1997.

[CvH91] David Chaum and Eugène van Heyst. Group signature. In D.W. Davies, editor, *Advances in Cryptology – EUROCRYPT '91*, pages 257–265. Springer-Verlag, LNCS 547, 1991.

[Dek00] Anthony H. Dekker. C3po: a tool for automatic sound cryptographic protocol analysis. In *13th IEEE Computer Security Foundations Workshop*, pages 77–87. IEEE CS Press, June 2000.

[DH99] Naganand Doraswamy and Dan Harkins. *IPSEC: The New Security Standard for the Internet, Intranets, and Virtual Private Networks*. Prentice Hall, 1999.

[DM00] G. Denker and J. Millen. Capsl integrated protocol environment. In *DISCEX 2000: Proceedings of the DARPA Information Survivability Conference and Exposition*, volume I, pages 207–221. IEEE CS Press, January 2000.

[DS81] Dorothy E. Denning and Giovanni Maria Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536, August 1981.

[DvOW92] Whitfield Diffie, Paul C. van Oorschot, and Michael J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes, and Cryptography*, 2:107–125, 1992.

[DY83] Danny Dolev and Andrew C. Yao. On the security of public-key protocols. *IEEE Transactions on Information Theory*, 2(29):198–208, March 1983.

[ES00] Neil Evans and Steve Schneider. Analysing time dependent security properties in CSP using PVS. In F. Cuppens, Y. Deswarte, D. Gollmann, and M. Waidner, editors, *Computer Security – ESORICS 2000*, pages 222–237. Springer-Verlag, LNCS 1895, October 2000.

[FHMV95] Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning About Knowledge*. MIT Press, 1995.

[Fre] Freedom. http://www.freedom.net/.

[GNY90] Li Gong, Roger Needham, and Raphael Yahalom. Reasoning about Belief in Cryptographic Protocols. In *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, pages 234–248. IEEE Computer Society Press, 1990.

[Gol92] Robert Goldblatt. *Logics of Time and Computation, 2^{nd} edition*, volume 7 of *CSLI Lecture Notes*. CSLI Publications, 1992.

[Gol96] Dieter Gollmann. What do we mean by entity authentication? In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 46–54. IEEE CS Press, May 1996.

[Gol00] Dieter Gollmann. On the verification of cryptographic protocols - a tale of two committees. In Steve Schneider and Peter Ryan, editors, *Electronic Notes in Theoretical Computer Science*, volume 32. Elsevier Science Publishers, 2000.

[GS98] Li Gong and Paul Syverson. Fail-stop protocols: An approach to designing secure protocols. In R. K. Iyer, M. Morganti, W. K. Fuchs, and V. Gligor, editors, *Dependable Computing for Critical Applications 5*, pages 79–100. IEEE Computer Society Press, 1998.

[Gut] Joshua D. Guttman. Security goals: Packet trajectories and strand spaces. This volume.

[HT96] N. Heintze and J. D. Tygar. A model for secure protocols and their composition. *IEEE Transactions on Software Engineering*, 22(1):16–30, January 1996.

[Jür00] Jan Jürjens. Bridging the gap: Formal vs. complexity-theoretical reasoning about cryptography, December 2000. Presentation at the Schloss Dagstuhl Seminar on Security through Analysis and Verification.

[Kai95] Rajashekar Kailar. Reasoning about accountability in protocols for electronic commerce. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 236–250. IEEE CS Press, May 1995.

[Kai96] Rajashekar Kailar. Accountability in electronic commerce protocols. *IEEE Transactions on Software Engineering*, 5(22), May 1996.

[Kem87] Richard A. Kemmerer. Using formal verification techniques to analyze cryptographic protocols. In *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, pages 134–139. IEEE CS Press, May 1987.

[KN98] Volker Kessler and Heike Neumann. A sound logic for analysing electronic commerce protocols. In J.-J. Quisquater, Y. Deswarte, C. Meadows, and D. Gollmann, editors, *Computer Security – ESORICS 98*, pages 345–360. Springer-Verlag, LNCS 1485, September 1998.

[KR00] Steve Kremer and Jean-François Raskin. A game approach to the verification of exchange protocols: Application to non-repudiation protocols. In P. Degano, editor, *First Workshop on Issues in the Theory of Security – WITS'00*, pages 93–98, July 2000.

[Lam78] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[Low96] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. *Software - Concepts and Tools*, 17:93–102, 1996.

[Low97] Gavin Lowe. A herarchy of authentication specifications. In *Proceedings of the 10th IEEE Computer Security Foundations Workshop (CSFW9)*, pages 31–43. IEEE Computer Society Press, June 1997.

[MCF87] Jonathan K. Millen, Sidney C. Clark, and Sheryl B. Freedman. The Interrogator: Protocol security analysis. *IEEE Transactions on Software Engineering*, 13(2):274–288, 1987.

[Mea94] Catherine Meadows. A model of computation for the NRL Protocol Analyzer. In *Proceedings of the 7th Computer Security Foundations Workshop*, pages 84–89. IEEE CS Press, June 1994.

[Mea96] Catherine Meadows. The NRL Protocol Analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, February 1996.

[Mea99a] C. Meadows. Analysis of the Internet Key Exchange protocol using the NRL Protocol Analyzer. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 1999.

[Mea99b] Catherine Meadows. A formal framework and evaluation method for network denial of service. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop (CSFW12)*, pages 4–13. IEEE CS Press, June 1999.

[Mea00a] Catherine Meadows. Extending formal cryptographic protocol analysis techniques for group protocols and low-level cryptographic primitives. In P. Degano, editor, *First Workshop on Issues in the Theory of Security – WITS'00*, pages 87–92, July 2000.

[Mea00b] Catherine Meadows. Open issues in formal methods for cryptographic protocol analysis. In *DISCEX 2000: Proceedings of the DARPA Information Survivability Conference and Exposition*, volume I, pages 237–250. IEEE Computer Society Press, January 2000.

[Mea01] Catherine Meadows. A cost-based framework for analysis of denial of service in networks. *Journal of Computer Security*, 2001. Forthcoming. A preliminary version of portions of this work appeared in [Mea99b].

[Men87] Elliott Mendelson. *Introduction to Mathematical Logic*. Wadsworth Publishing Co., 1987.

[Mil] J. Millen. Capsl Web site. `www.csl.sri.com/~millen/capsl`.

[Mil84] Jonathan K. Millen. The Interrogator: A tool for cryptographic protocol security. In *Proceedings of the 1984 IEEE Symposium on Security and Privacy*, pages 134–141, Oakland, CA, April 1984. IEEE Computer Society Press.

[Mos86] Paul K. Moser, editor. *Empirical Knowledge: Readings in Contemporary Epistemology*. Rowman & Littlefield, 1986.

[MR00] Jon Millen and Harald Rueß. Protocol-independent secrecy. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 110–119, May 2000.

[MS98] C. Meadows and P. Syverson. A formal specification of requirements for payment trnasactions in the SET protocol. In R. Hirschfeld, editor, *Financial Cryptography, FC'98*, pages 122–140. Springer-Verlag, LNCS 1465, 1998.

[MvOV97] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[Nes90] D.M. Nessett. A critique of the burrows, abadi, and needham logic. *Operating Systems Review*, 24(2):35–38, April 1990.

[NS78] R.M. Needham and M.D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.

[NS93] B. Clifford Neuman and Stuart G. Stubblebine. A Note on the Use of Timestamps as Nonces. *Operating Systems Review*, 27(2):10–14, April 1993.

[Oni] Onion routing. http://www.onion-router.net/.

[Pau97] Lawrence C. Paulson. Mechanized proofs for a recursive authentication protocol. In *Proceedings of the 10th IEEE Computer Security Foundations Workshop (CSFW10)*, pages 84–94. IEEE CS Press, June 1997.

[PS00] Adrian Perrig and Dawn Song. Looking for diamonds in the desert — extending automatic protocol generation to three-party authentication and key agreement. In *13th IEEE Computer Security Foundations Workshop*, pages 64–76. IEEE CS Press, June 2000.

[Ros96] A.W. Roscoe. Intensional specification of security protocols. In *Proceedings of the 9th IEEE Computer Security Foundations Workshop (CSFW9)*, pages 28–36. IEEE CS Press, June 1996.

[Sat89] M. Satyanarayanan. Integrating security in a large distributed system. *ACM Transactions on Computer Systems*, 15(3):247–280, August 1989.

[Sch96] Bruce Schneier. *Applied Cryptography, Second Edition: Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, 1996.

[SET97] Secure Electronic Transaction Specification, Version 1.0, May 1997. http://www.visa.com/set/.

[SM93] P. Syverson and C. Meadows. A logical language for specifying cryptographic protocol requirements. In *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, pages 165–177. IEEE CS Press, May 1993.

[SM94] P. Syverson and C. Meadows. Formal requirements for key distribution protocols. In A. De Santis, editor, *Advances in Cryptology — EUROCRYPT '94*, pages 32–331. Springer-Verlag, LNCS 950, 1994.

[SM96] P. Syverson and C. Meadows. A formal language for cryptographic protocol requirements. *Designs, Codes, and Cryptography*, 7(1 and 2):27–59, January 1996.

[Sne91] Einar Snekkenes. Exploring the BAN approach to protocol analysis. In *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, pages 171–181. IEEE CS Press, May 1991.

[Sne92] Einar Snekkenes. Roles in cryptographic protocols. In *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, pages 105–119. IEEE CS Press, May 1992.

[Son99] Dawn Song. Athena: a new efficient automatic checker for security protocol analysis. In *Proceedings of the Twelth IEEE Computer Security Foundations Workshop*, pages 192–202, Mordano, Italy, June 1999. IEEE Computer Society Press.

[SS96] Steve Schneider and Abraham Sidiropoulos. CSP and anonymity. In E. Bertino, H. Kurth, G. Martella, and E. Montolivio, editors, *Computer Security – ESORICS 96*, pages 198–218. Springer Verlag, LNCS 1146, 1996.

[SS99] Paul Syverson and Stuart Stubblebine. Group principals and the formalization of anonymity. In J.M. Wing, J. Woodcock, and J. Davies, editors, *FM'99 – Formal Methods, Vol. I*, pages 814–833. Springer-Verlag, LNCS 1708, 1999.

[Sti95] Douglas R. Stinson. *Cryptography: Theory and Practice.* CRC Press, 1995.

[SvO94] Paul F. Syverson and Paul C. van Oorschot. On unifying some cryptographic protocols. In *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, pages 14–28. IEEE CS Press, May 1994.

[SvO96] Paul F. Syverson and Paul C. van Oorschot. A unified cryptographic protocol logic. NRL Publication 5540-227, Naval Research Lab, 1996.

[SW96] Stuart Stubblebine and Rebecca Wright. An authentication logic supporting synchronization, revocation, and recency. In *3rd ACM Conference on Computer and Communications Security (CCS'96)*, pages 95–105. ACM Press, March 1996.

[Syv91] Paul F. Syverson. The use of logic in the analysis of cryptographic protocols. In *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, pages 156–170. IEEE CS Press, May 1991.

[Syv92] Paul F. Syverson. Knowledge, belief, and semantics in the analysis of cryptographic protocols. *Journal of Computer Security*, 1(3,4):317–334, 1992.

[Syv93a] Paul F. Syverson. Adding time to a logic of authentication. In *Proceedings of the First ACM Conference on Computer and Communications Security*, pages 97–101, November, 1993. ACM Press.

[Syv93b] Paul F. Syverson. On Key Distribution Protocols for Repeated Authentication. *Operating Systems Review*, 27(4):24–30, October 1993.

[Syv94] Paul Syverson. A taxonomy of replay attacks. In *Proceedings of the Computer Security Foundations Workshop (CSFW7)*, pages 187–191. IEEE CS Press, June 1994.

[Syv96] Paul Syverson. Limitations on design principles for public key protocols. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 62–72. IEEE Computer Society Press, May 1996.

[Syv98] Paul F. Syverson. Relating two models of computation for security protocols. In *Workshop on Formal Methods and Security Protocols*, Indianapolis, Indiana, June 1998. Available at http://www.cs.bell-labs.com/who/nch/fmsp/program.html.

[Syv00] Paul F. Syverson. Towards a strand semantics for authentication logic. *Electronic Notes in Theoretical Computer Science*, 20, 2000. Proceedings of MFPS XV (S. Brookes, A. Jung, M. Mislove and A. Scedrov, eds.), New Orleans, LA, April 1999.

[THG97] F. Javier Thayer Fábrega, Jonathan C. Herzog, and Joshua D. Guttman. Strand spaces. Technical report, The MITRE Corporation, November 1997.

[THG98a] F. Javier Thayer Fábrega, Jonathan C. Herzog, and Joshua D. Guttman. Honest ideals on strand spaces. In *Proceedings of the 1998 IEEE Computer Security Foundations Workshop — CSFW'11*, pages 66–77. IEEE Computer Society Press, 1998.

[THG98b] F. Javier Thayer Fábrega, Jonathan C. Herzog, and Joshua D. Guttman. Strand spaces: Why is a security protocol correct? In *Proceedings of the 1998 IEEE Symposium on Security and Privacy*, pages 160–171, Oakland, CA, May 1998. IEEE Computer Society Press.

[THG99a] F. Javier Thayer Fábrega, Jonathan C. Herzog, and Joshua D. Guttman. Mixed strand spaces. In P. Syverson, editor, *Proceedings of the 12th IEEE Computer Security Foundations Workshop — CSFW'99*, pages 72–82, Mordano, Italy, June 1999. IEEE Computer Society Press.

[THG99b] F. Javier Thayer Fábrega, Jonathan C. Herzog, and Joshua D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7(2,3):191–230, 1999.

[Tou92] Marie-Jeanne Toussaint. Separating the specification and implementation phases in cryptology. In Y. Deswarte, G. Eizenberg, and J.-J. Quisquater, editors, *Computer Security – ESORICS 92*, pages 77–102. Springer-Verlag, LNCS 648, November 1992.

[Tut] Mark Tuttle. Flaming in Franconia. Remarks made in a panel discussion on the use of formal methods in the analysis of cryptographic protocols at the IEEE Computer Security Foundations Workshop in Franconia, New Hampshire, June 1992.

[vO93] Paul C. van Oorschot. Extending cryptographic logics of belief to key agreement protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 233–243. ACM Press, November 1993.

[WK96] Gabriele Wedel and Volker Kessler. Formal semantics for authentication logics. In E. Bertino, H. Kurth, G. Martella, and E. Montolivo, editors, *Computer Security – ESORICS 96*, pages 219–241. Springer-Verlag, LNCS 1146, September 1996.

[Yah93] Raphael Yahalom. Optimality of asynchronous two-party secure data-exchange protocol. *Journal of Computer Security*, 2(2–3):191–209, 1993.

[ZG98] Jianying Zhou and Dieter Gollmann. Towards verification of non-repudiation protocols. In T. Vickers J. Grundy, M. Schwenke, editor, *International Refinement Workshop and Formal Methods Pacific 1998*, pages 370–380. Springer-Verlag, 1998.