DNSSEC: Verification, Validation and Proposal for Enhancement

A thesis submitted during 2017 to the University of Hyderabad

in partial fulfillment of the requirements for the award of

Doctor of Philosophy

in

Computer Science

by

Kollapalli Ramesh Babu

Reg. No. 08MCPC09

School of Computer and Information Sciences

University of Hyderabad

P.O. Central University, Gachibowli

Hyderabad - 500 046, India.

CERTIFICATE

This is to certify that the thesis entitled **"DNSSEC: Verification, Validation and Proposal for Enhancement"** submitted by **Kollapalli Ramesh Babu** bearing registration number **08MCPC09** in partial fulfillment of the requirements for the award of **Doctor of Philosophy** in **Computer Science** is a bonafide work carried out by him under our supervision and guidance.

The thesis is free from plagiarism and has not been submitted previously in part or in full to this or any other University or Institution for the award of any degree or diploma.

Parts of this thesis have been presented in the following International Conferences:

1. Kollapalli Ramesh Babu, Vineet Padmanabhan and Wilson Naik Bhukya, Reasoning about DNSSEC, MIWAI 2011: pp. 75 - 86.
2. Vineet Padmanabhan, Abdul Sattar, Guido Governatori, and Kollapalli Ramesh Babu, Incorporating temporal planning within a BDI architecture, IICAI 2011: pp. 1618 - 1636.
3. Kollapalli Ramesh Babu and Vineet Padmanabhan, Automated validation of DNSSEC, at International Conference on Computing Analytics and Networking (ICCAN 2017).
4. Kollapalli Ramesh Babu and Vineet Padmanabhan, BDI based performance enhancement in DNSSEC, at ICDCIT 2018.

Further, the student has passed the following courses towards fulfillment of coursework requirement for Ph.D.

S.No	Code	Course Name	Credits	Result
1	CS801	Data Structures and Algorithms	4	Pass
2	CS802	Operating Systems and Programming	4	Pass
3	AI852	Learning and Reasoning	4	Pass
4	IT811	Secure Computing	4	Pass

Supervisor

Prof. Vineet C.P. Nair

Dean

Prof. Arun Agarwal

DECLARATION

I, **K. RAMESH BABU**, hereby declare that this thesis entitled "**DNSSEC: Verification, Validation and Proposal for Enhancement**" submitted by me under the guidance and supervision of **Prof. VINEET C.P. NAIR** is a bonafide research work. I also declare that it has not been submitted previously in part or in full to this University or any other University or Institution for the award of any degree or diploma.

Date:

Name: **K. RAMESH BABU**

Signature of the Student

Reg. No. **08MCPC09**

Abstract

It is very common these days for any user of the internet to type a hostname like `uohyd.ac.in` into a web browser like google to retrieve relevant information about the domain from the world wide web. The translation of human readable hostnames such as `uohyd.ac.in` into Internet protocol addresses like 183.82.173.128 is done with the help of Domain Name System (DNS). The Domain Name System has a hierarchical distributed database structure and does not include any security as it was designed to be a public database. Almost in every interaction we have with the internet like Email, Web services etc. we make use of DNS and therefore there is a strong demand for securing the communication within the DNS system. DNSSEC is a security protocol designed to authenticate and protect the integrity of DNS through the use of public key-based digital signatures. Even though the security protocols are designed and implemented with utmost care, it is proved that many of the security protocols which have been in use for many years have logical flaws hidden inside. In this thesis the first issue we address is with regard to providing a formal framework based on SVO logic for representing and reasoning about DNSSEC so as to validate it against any logical flaws. Formal Analysis of DNSSEC was carried out by verifying *origin authentication* and *data integrity* with the help of message passing examples. By making use of the inference rules as well as the axioms of SVO logic we were able to successfully derive the proof for data origin authentication and data integrity of the DNSSEC protocol. The second issue is related to the validation of DNSSEC protocol wherein we had to translate the specification in SVO logic into a machine understandable format and this was made possible through the AVISPA tool and HLPSL converter. The simulation results ensured that the DNSSEC protocol is safe. The third issue is related to the performance enhancement in DNSSEC where the challenge was not only in designing a provable protocol but also to get one that works with *time limits*. We had to adopt an agent-oriented approach to solve this issue and proposed a temporal planning model through which an improvement in the DNSSEC performance in terms of construction of *chain of trust* was achieved.

Acknowledgments

I would like to express my sincere thanks to **Prof. Vineet C.P. Nair**, my research supervisor, for his valuable suggestions and keen personal interest throughout the course of my research. He encouraged, supported, corrected and guided me during the work. This project has been a great learning and growing experience for me. It is a good time of work done under his guidance and a great opportunity to learn many things. I am extremely grateful to **Prof. Arun Agarwal**, Dean of SCIS, for providing excellent computing facilities and a disciplined atmosphere for doing my research.

I am indebted to **Prof. C. R. Rao** and **Prof. S. K. Udgata**, the members of my review committee, who made me strong enough to complete my experiments by providing their time, effort and feedbacks during my periodical assesments.

I would like to sincerely thank **Prof. Hrishikesh Mohanty** for his motivation, suggestions and support in all respect.

And I would also like to thank **Prof. Girija**, **Prof. K. N. Murthy**, **Prof. S Bapi Raju**, **Prof. Chakravarthy**, **Prof. Atul Negi**, **Prof. Rajeev Wankar**, **Prof. Alok Singh**, **Dr. S Durga Bhavani**, **Dr. Anupama Potluri**, **Dr. Rajendra Prasad Lal** and other Teaching and Non-teaching staff of SCIS.

And my special thanks to my parents, family members, friends, relatives, colleagues and co-researchers in **UOH** who supported me with their moral support.

Contents

List of Figures **viii**

List of Tables **x**

1 Introduction **1**

1.1 Research Challenges 3

1.2 Proposed Solutions 6

1.3 Thesis Contributions 9

1.4 Thesis Organization 11

2 Preliminaries, Foundational Concepts and Related Work **13**

2.1 DNS Infrastructure 14

2.1.1 The Domain name space and resource records 14

2.1.2 Name servers 15

2.1.3 Resolvers 16

2.1.4 Work flow of DNS 17

2.2 Attacks on DNS 19

2.2.1 Attacks on DNS Protocols 20

2.2.2 Attacks on DNS in Real time Scenario 20

2.3 DNSSEC 21

2.4 Integrity 22

2.4.1 Types of Data Integrity 22

2.4.2 Causes of Integrity Violations 23

2.4.3 Methods to Verify Data Integrity 24

2.5 Authentication 25
2.5.1 Types of Authentication 26
2.5.2 Factors used in Authentication 26
2.5.3 Methods to Verify Authentication 27
2.6 Formal Methods for Security Protocols 28
2.6.1 Classification of Formal Methods 29
2.7 Modal Logics to Verify Security Protocols 31
2.7.1 Burrows Abadi Needham (BAN) Logic 33
2.7.2 Gong Needham Yahalom (GNY) Logic 39
2.7.3 Abadi Tuttle (AT) Logic 40
2.7.4 Van Oorschot (VO) Logic 41
2.7.5 SyVerson-van Oorschot (SVO) Logic 41
2.8 Automation Tools to Validate Security Protocols 41
2.8.1 NRL Protocol Analyzer 42
2.8.2 Failure Divergence's Refinement (FDR) Checker 43
2.8.3 Longley-Rigby Tool 44
2.8.4 ProVerif 44
2.8.5 Scyther Tool 45
2.8.6 AVISPA Tool 46
2.9 Performance Enhancement Methods
for DNSSEC Protocol 47
2.9.1 DHT-based Architectures 47
2.9.2 PREFETCHing 48
2.9.3 Optimizing Negative Caching 49
2.9.4 Negotiating DNSSEC Algorithms 50
2.9.5 An Advanced Client Based DNSSEC Validation 50
2.10 Summary 51

3 Formal Analysis of DNSSEC Protocol 52
3.1 Domain Name System Security Extensions (DNSSEC) Protocol . . 54
3.1.1 DNSSEC Work Flow 56

3.1.2 Origin Authentication and Data Integrity 59
3.2 Formal Representation of DNSSEC Protocol 60
3.2.1 Constructs used in DNSSEC protocol representation 61
3.2.2 DNSSEC Protocol in (A - B) Notation 61
3.3 SVO Logic for DNSSEC Protocol 62
3.3.1 SVO Inference Rules 64
3.3.2 SVO Axioms for Reasoning about DNSSEC 64
3.3.3 Extended SVO logic for Reasoning about DNSSEC 67
3.4 Formal Analysis of DNSSEC 68
3.4.1 SVO Logic Procedure to Analyze Protocol 68
3.4.2 Protocol Analysis 70
3.5 Summary 74

4 Validating DNSSEC protocol **75**
4.1 Security Protocol Verification Tool: AVISPA 75
4.1.1 On-the-Fly Model-Checker (OFMC) 77
4.1.2 SAT-based Model-Checker (SATMC) 78
4.1.3 Constraint-Logic-based Attack Searcher (CL-AtSe) 78
4.1.4 Tree Automata based on Automatic Approximations for the Analysis of Security Protocols (TA4SP) 79
4.2 An Overview of DNSSEC Specification Process 80
4.3 AVISPA and HLPSL 82
4.4 Validating DNSSEC using AVISPA 85
4.5 Results 90
4.5.1 Results with On-the-fly Model-Checker (OFMC) as back-end 91
4.5.2 Results using Constraint-Logic-based Attack Searcher (CL-AtSe) as a back-end 94
4.6 Summary 97

5 Analysing DNSSEC as a Planning problem in Multi-Agent Systems **98**

5.1 Introduction to BDI 100
5.2 BDI Architecture 102
5.3 Classical Planning Cycle 103
5.4 Temporal Planning 105
5.5 Generation of Temporal Plan within BDI 106
5.6 Temporal Planning with BDI - A case study 111
5.6.1 Solution 114
5.7 DNSSEC with Temporal BDI 127
5.8 Performance Enhancement Method for DNSSEC with Temporal BDI 128
5.9 Performance Study with Example 130
5.9.1 DNS Request Resolution and Verification Process 131
5.9.2 Belief Base Updation Process 134
5.9.3 Plan Generation Process to Construct Chain of Trust 135
5.10 Reasoning about DNSSEC with temporal BDI in (Alice - Bob) notation 137
5.10.1 The Reasoning Process 138
5.11 Results 139
5.12 Summary 145

6 Conclusions and Future Work **147**

References **150**

List of Publications **159**

List of Figures

2.1 DNS Resource Record format 14

2.2 DNS Query format 16

2.3 DNS Response format 16

2.4 DNS query resolution in Recursive manner 17

2.5 DNS query resolution in Iterative manner 18

2.6 Wide mouthed frog protocol 37

2.7 Protocol analysis with BAN logic 39

3.1 A sample scenario of DNS 58

4.1 Architecture of AVISPA Tool 76

4.2 OFMC Summary for DNSSEC Protocol with one session 91

4.3 OFMC simulation for DNSSEC Protocol with one session 91

4.4 OFMC simulation for DNSSEC Protocol with one session including intruder 92

4.5 OFMC Summary for DNSSEC Protocol with two sessions 92

4.6 OFMC simulation for DNSSEC Protocol with two sessions 93

4.7 OFMC simulation for DNSSEC Protocol with two sessions including intruder 93

4.8 CL-AtSe Summary for DNSSEC Protocol with one session 94

4.9 CL-AtSe simulation for DNSSEC Protocol with One session 94

4.10 CL-AtSe simulation for DNSSEC Protocol with one session including intruder 95

4.11 CL-AtSe Summary for DNSSEC Protocol with two sessions 95

4.12 CL-AtSe simulation for DNSSEC Protocol with two sessions 96
4.13 CL-AtSe simulation for DNSSEC Protocol with two sessions including intruder 96

5.1 BDI Architecture. 102
5.2 Modified BDI Architecture. 107
5.3 A Flow Diagram for Modified BDI Architecture. 111
5.4 Complete solution of the travel problem 126
5.5 DNSSEC Within BDI 129
5.6 DNSSEC overhead 140
5.7 Chain of trust construction time 141
5.8 Number of DS requests to construct chain of trust 142
5.9 DNSSEC Authentication Chain for www.iith.ac.in 143
5.10 DNSSEC Authentication Chain for www.upsc.gov.in 144

List of Tables

2.1 Types of Resource Records 15

2.2 Categorization of Tools for cryptographic protocol analysis 47

2.3 Characteristic and Performance Comparison among DNS, NCDO and DNSSEC 49

3.1 Constructs used in DNSSEC protocol representation 61

5.1 An example of classical planning 104

5.2 Belief-base 112

5.3 Action-base 112

5.4 Plan-library 113

5.5 Belief-base of DNSSEC. 130

5.6 Action-base of DNSSEC. 130

5.7 Plan generated by BDI to resolve DNS query 134

5.8 Initial belief-base 134

5.9 Belief-base after step 1 to 4 134

5.10 Belief-base after step 5 to 8 135

5.11 Belief-base after step 9 to 12 135

5.12 Belief-base after step 13 to 16 135

5.13 Belief-base after step 17 to 20 135

5.14 Belief-base after step 21 to 24 135

5.15 Plan generated by BDI to construct chain of trust 136

5.16 Belief-base after execution of chain of trust plan 136

Chapter 1

Introduction

In this age of Internet, it is well known that, every system that is connected via a computer network is assigned with a unique address called the Internet Protocol(IP) Address, to identify and access different systems. These addresses are numerical labels assigned to each device connected to a computer network and makes use of the *Internet Protocol* (IP) for communication. Humans make use of these networked systems and find it easy to locate different websites by typing in different domain names rather than the numerical labels. The `domain name system` (DNS) [67, 66, 68] maps the domain names that people have used to locate a website to the IP address of the computer that is used to locate a website. For example, if someone types `uohyd.ac.in` into a web browser like google, a server behind the scenes will map that name to the IP address 206.19.49.149.

The DNS has a distributed database structure and does not include any security as it was designed to be a public database. This has lead to security vulnerabilities wherein the DNS process of *looking a site up on the internet* for the user could be hijacked and the user could be send to the hijacker's own deceptive web site for account and password collection. In other words an attacker can fool a cache into accepting false DNS data. It is also the case that various man-in-the-middle kind of attacks are possible. To overcome these vulnerabilities a technology called DNS Security Extensions (DNSSEC) was introduced that add security to the Domain Name System (DNS) protocol by enabling DNS responses to be validated. With

DNSSEC, the DNS protocol is much less susceptible to certain types of attacks, particularly DNS spoofing attacks. The main design motive behind DNSSEC is not to end the various kinds of attacks but to make the attacks detectable by the end-user. This in a way saves users from doing online transactions on the wrong server even if a secured connection is used and the address in the browser looks correct. DNSSEC provides security to DNS system by adding four new resource records (RRs) and *constructing a chain of trust*. DNSSEC protocol provides two types of security services, namely, *data integrity* and *data authentication*. Data integrity ensures that the `received` data is the same as that of the `sent` one. Data authentication, on the other hand, ensures the genuineness of a sender server.

The challenge is not only in designing a provable protocol but also to get one that works in a desired time limit. In order to construct a chain of trust among servers lying between network edge server and a domain server, a client of an edge server sends a request to its immediate parent server and in return the latter sends a similar request to its parent until the request reaches the domain root-server where the communicating client-server genuineness can be established. This process establishes a chain of trust among the servers lying between the communicating clients. Unless this chain of trust is established the issues of authentication and data integrity cannot be resolved. Data received through an unauthenticated communication is discarded. The process of establishing a chain of trust is fully dependent on the DNSSEC protocol.

Like any other protocol, the efficiency of DNSSEC is due to its specification and concreteness. A protocol is a concrete one when its functionalities are provable. In the literature it is often referred to as proving the correctness of a protocol. To achieve the correctness of a protocol, formal analysis of the protocol needs to be carried out. This is the *first issue* this thesis addresses. In this thesis we provide a formal analysis of DNSSEC in the background of Modal Logic which to the best of our knowledge have not been attempted before. Before making a software protocol operational, it is extremely important to verify its specification. In the case of a protocol with a formal specification, its functionalities need to be proved to ensure a desired goal (protocol functionality) is reachable from the given ground truth (logic based specification) of the protocol. In this case, DNSSEC's authenticity as well as data integrity are to be proved from its formal specification. Logic based verification mechanism verifies the given security protocols at a high level of abstraction, though in general attacks on security protocols are nonintuitive and subtle in nature. In order to validate the security functionalities of a given security protocol at low-level of abstraction, we need to take the help of automatic validation tools. Keeping this need in view, the *second issue* addressed in this thesis is about the automatic validation of DNSSEC protocol. In this thesis we make use of the AVISPA tool for validating the DNSSEC protocol. The *third issue* to be addressed of a protocol is its timeliness. Particularly in the case of an authentication protocol like DNSSEC, timeliness is the demanding one because it is not desirable for a user needing Internet services to wait too long for authentication. To this end we adopt an agent-oriented approach to solve this issue and propose a temporal planning model through which an improvement in the DNSSEC performance in terms of construction of chain of trust is achieved. In a nutshell, the three issues addressed in this thesis with regard to DNSSEC protocol are:

1. Formal analysis of DNSSEC protocol using SVO logic framework.

2. Automatic validation of DNSSEC protocol using the AVISPA tool.

3. Performance enhancement of DNSSEC using temporal BDI agent model.

On identification of the above three issues, in the next section we present the research challenges this thesis has taken up.

1.1 Research Challenges

Designing authentication protocols like DNSSEC is always a challenging task because these protocols are often error prone. The best example of such a flawed protocol is the Needham-Schroeder public key protocol [69] . In this simple public key protocol a flaw was found after seventeen years of its publication by *Lowe* using model checker FDR (Failures Divergences Refinement) [53]. Another *logical flaw* was discovered in the renegotiation feature of the widely used TLS (Transport Layer Security) protocol, thirteen years after the first version of the protocol was published. Several type-confusion attacks against the Group Domain of Interpretation Protocol [57] were identified by using NRL protocol analyzer. In a similar manner, there are number of instances in the literature where logical flaws in security protocols were discovered only after the protocol was in use for many years.

In the previous section we have identified three research issues related to DNSSEC protocol. The *first issue* was related to the formal analysis of the DNSSEC protocol. Formal analysis for security protocols is mainly done to unveil the *logical flaws* and/or to prove correctness of a given protocol. The challenges involved in formal analysis of a security protocol includes that of comprehending a security protocol thoroughly in terms of initial conditions, assumptions, constraints, messages exchanged etc. since these factors play a vital role in deriving the goals of a protocol and proving the correctness of a protocol. Misunderstanding some concept/part of a protocol will definitely lead to wrong conclusions which in turn could result in either proving a correct protocol wrong or a wrong protocol correct. There are many instances in the literature where incorrect protocols are proved formally correct and vice versa. This happens mainly due to wrong interpretation or lack of thorough understanding of a protocol.

The *second issue* we mentioned was related to the development of an automated tool for validation of the DNSSEC protocol. Basically, the security protocols are very complex in nature. Most of the time manual validation of such protocols clearly outpaces human capacity. This is because the vulnerabilities are not directly related to cryptographic techniques implemented by the security protocols. Some of the vulnerabilities are related to bad interactions among multiple parallel sessions of the same protocol and are very subtle in nature. For instance, an intruder can make use of a *protocol agent* as an oracle to obtain some crucial information which the intruder could not generate on its own. The intruder will make use of the information thus obtained to forge new messages and gets it injected into other parallel sessions. To identify these type of attacks we need the assistance of automation tools through which we can test the protocol with more number of parallel sessions and other constraints. Research challenge in this case is writing the protocol specification in a language supported by the tool without any errors which in turn also needs a thorough understanding of the protocol. Another challenge is selecting the most appropriate tool that matches our requirements and should be the best among the existing ones, which can handle the protocol under consideration.

The *third issue* we mentioned in the previous section is with regard to improving the performance of the DNSSEC protocol. DNSSEC protocol protects the DNS system by verifying the authentication and integrity of DNS responses. DNSSEC protocol is able to achieve this authentication and integrity by imposing some overhead on the present DNS system. What these overheads mean from the clients perspective is that of acquiring additional information to authenticate the received data. In order to get additional information, the client has to send more number of queries and receive response from servers as well as perform additional operations like storing, comparing etc. The overhead from servers perspective is in the maintenance of additional information regarding all of its immediate descendant servers which are responsible for sub-domains (zones). The overhead from networks point of view is in the handling of additional traffic due to extra requests/ responses that are sent/ received by clients and servers. Research challenge in this third issue has to do with the minimisation of the overhead wherever it is possible so that the overall performance of the system can be improved. Research challenge in specific is in the reduction of the number of chain of trust constructions as far as possible or else reducing the time to construct the chain of trust in all possible ways. Another challenge is to ensure that the authentication process takes place in a `timed` manner.

1.2 Proposed Solutions

The first problem that we addressed in this thesis is that of formally analysing the DNSSEC protocol. Formal analysis can be interpreted as the usage of ”mathematical techniques” to ensure that the design or implementation of the security protocol confirms to the specified security goals. Mathematical techniques or formal methods that are used to verify security protocols are broadly classified into three categories: The first one is based on *Model checking* methods whereas the second one is based on the method of *Theorem provers*, and the third one is based on methods that make use of *Logical inference* [24]. Methods based on model checking explore all states and transitions in the model by using smart and domain specific techniques. Methods based on theorem provers search at a higher level of abstraction for chains of logic that constitute compelling proof that a particular property always holds. Methods based on logical inference uses formal version of mathematical reasoning to reason about the model in hand and is usually driven by the users understanding of the intricacies involved in the model.

In this thesis we adopted the technique based on logical inference to specify and verify the DNSSEC protocol. In this logical inference technique we can make use of different logics like BAN logics to derive the goals of the given security protocol. BAN logic [24] is an ancestor of other logics like AT [3], GNY [45] and SVO [79]. In this thesis we propose to use the SVO logic [79] for formal reasoning and verification of the DNSSEC protocol. SVO logic was designed to capture the features and extensions of four variants of logics, namely `BAN, GNY, AT, VO`, in a single unified framework. *SVO logic* was designed in such a way that, it is simple to use and more expressive than any of the logics from which it is derived; more details are discussed in section 2.7. Logics like SVO are particularly suitable for protocol verification because they are comparatively simple and effective.

In order to reason about an authentication protocol like DNSSEC using *SVO* logic we need to follow five step process. In the first step, we write initial conditions of the protocol, which includes the beliefs held by the principals and the possession of keys or any other prior assumptions the principals might be holding. In the second step we write the messages that are exchanged between principals which include the received messages, that is, the messages are that not lost in transit. In the third step we assert that the messages that are received by the principals are comprehended (eg. decryption of received message by using a key that is known already). In the fourth step we write the inferred messages by principals with the received messages and the knowledge that principals initially have. In the fifth step we use axioms and premises to derive the goals of the protocol. If we succeed in deriving the specified goals of the protocol then we can say that the protocol is working correctly according to the specification, else we can have suspicion about the protocol's correctness.

One has to be very careful while writing the initial conditions i.e., formalization of the protocol messages. Mistakes committed while formalizing the protocol messages and initial conditions render the verification process worthless. Detailed solution of formal analysis of DNSSEC protocol is discussed in chapter 3. As the complexity of a security protocol grows, it becomes a very difficult and tedious task to verify the correctness manually. To cope up with the complexity and ensuring of security functions of a given protocol, automatic validation tools [19, 32] are necessary. Automated validation tools for security protocols also play vital role in analyzing the working procedure of a protocol, specifically in parallel sessions and finding subtle errors.

In this thesis we propose to use the AVSPA tool to verify the DNSSEC security protocol. Automated Validation of Internet Security Protocols and Applications (AVISPA) [9, 80] is an automatic validation tool used to verify the correctness of security properties of Internet security protocols. AVISPA uses a High Level Protocol Specification Language (HLPSL) [28, 11] in which the protocol to be verified is specified. HLPSL is a very expressive and intuitive language used to formalise any authentication protocols so that the AVISPA tool can be used for verifying the correctness. More details are discussed in section 4.3. HLPSL is basically a role based language, which means that each participant in a security protocol is represented as one role in HLPSL specification. In general, HLPSL specification of a protocol consists of one or more basic role definitions, followed by the definition of composite role. In a composite role, one or more basic roles are combined together so that they can be executed together, usually, in parallel. After composite role, an environment role is defined by instantiating the composition role to create one or more sessions. Finally, we specify the security parameters as goals that we want to verify by the back end tools. AVISPA currently supports four backend tools to validate the protocol. They are On-the-fly Model-Checker (OFMC) [16], SAT-based Model-Checker (SATMC) [8], Constraint-Logic-based Attack Searcher (CL-AtSe) [83] and Tree Automata based on Automatic Approximations for the Analysis of Security Protocols(TA4SP) [20]. Further details are discussed in Section 4.1.

The AVISPA tool accepts input specification in HLPSL and translates HLPSL specifications into Intermediate Format (IF). IF is an intermediate and independent form of a given specification, which can be later used by back-end tools for analysis. The IF format of a protocol is executed over finite number of times or entirely if no loop exist. Finally, the execution may end-up with either by an identification of an attack on a given protocol or the protocol is proved as safe over a given number of sessions.We have represented the DNSSEC protocol in HLPSL and ensured the correctness of both the security parameters, i.e., data integrity and data origin authentication. The specification and validation details of DNSSEC protocol are explained in the later Chapters.

We are aware that the authentication, specifically in networking, is temporal. It means that the session keys, One time passwords (OTPs) etc. which are used for authentication are valid only for a limited period of time. In order to utilise the authentication information more effectively and improve the performance of DNSSEC protocol, in this thesis we have proposed DNSSEC with temporal BDI. In a BDI system [35, 72], Beliefs are basically facts about the environment in which a system exists, Desires are the goals the system want to achieve and Intentions are the goals which are currently under consideration. Preparation of a plan to achieve the goal is the important task in BDI system. Method of plan generation depends on the problem to be solved. In general a plan is a sequence of actions to be executed for achieving the goal. In order to find plans for the problems that deal with time, temporal planning technique is used. In real time problems, planning with time is very important as it means that performing a poor action within a specified time is preferable to performing a planned action which is too late. The planner has to generate a plan that is optimal and satisfies user constraints.

Our aim in this thesis is to extend the basic BDI architecture [21] and the classical planning life cycle to incorporate *temporal planning*. After proposing and analysing temporal planing in BDI system, we have utilised this technique for DNSSEC protocol to improve the performance. DNSSEC protocol uses temporal planning method to establish *chain of trust*. It is obvious and apparent that many requests and responses are generated and transmitted over a network during construction of chain of trust. In order to avoid unnecessary construction of chain of trusts, authentication data obtained during construction of chain of trust are saved as beliefs in BDI system for a valid time period. Authentication data saved in BDI system can be used by the client of DNSSEC protocol to answer the relevant queries without reconstruction of the chain of trust and thereby improve the performance of DNSSEC protocol as well as reducing Internet traffic. This will also reduce the response time of the DNS query. The details regarding the usage of BDI-based temporal planning to improve the performance of the DNSSEC protocol are discussed in Chapter 5.

1.3 Thesis Contributions

As mentioned in the previous section, In this thesis we provide reasoning about the security mechanisms involved in the DNSSEC protocol. DNSSEC protocol uses *chain of trust* technique to ensure *data integrity* and *origin authentication* of received data. In order to ensure that the process of chain of trust is formally correct, SVO logic based specification and verification method is proposed. *The results related to this work has been published in the form of a conference pa-*

per titled `Reasoning about DNSSEC` *in the Fifth Multidisciplinary International Conference on Artificial Intelligence (MIWAI), 2011.*

In order to validate the DNSSEC protocol at a low level of abstraction in-terms of parallel sessions and other constraints we demonstrate how the AVISPA (Automatic Verification of Internet Security Protocols and Applications) tool can be used. AVISPA tool is an automatic verification tool for internet security protocols. In the process of validation, first we have represented the DNSSEC protocol using HLPSL (High Level Protocol Specification Language) language. AVISPA translates the HLPSL specification into IF (Intermediate Form) format. The IF form is then analyzed by using various model checkers like SATMC, OFMC, CL-Atse, TA4SL to ensure security parameters. *The results related to this work has been accepted at the International Conference on Computing Analytics and Networking (ICCAN) - 2017 with the title* `Automated Validation of DNSSEC` *.*

The standard BDI architecture is extended to incorporate temporal planning. Generating a plan for a given problem is one of the important procedures in BDI system. Certain problems need only the plan, whereas other problems need a plan which can complete the task in a given time. When time plays an important role we need to go for techniques that can account for temporal planning in order to generate a plan. We propose a temporal planning framework for the standard BDI architecture. *The results of this work is published as* `Incorporating Temporal Planning Within a BDI Architecture` *at the 5th Indian International Conference on Artificial Intelligence (IICAI 2011).*

The proposed temporal planning BDI model is extended to apply in the case of the DNSSEC protocol so as to improve the performance of the authentication mechanism in DNSSEC. Temporal planning technique is further used to establish the chain of trust in DNSSEC. BDI-based architecture is used to save authentication information for a valid amount of time so that when a request comes the information related to the request is first checked with the saved authentication information. *The results related to this work has been accepted at the 14th International Conference on Distributed Computing and Internet Technology (ICDCIT-*

2018 with the tile `BDI-based Performance Enhancement in DNSSEC`*).*

1.4 Thesis Organization

Apart from the first chapter which has an introductory tone this thesis has five more chapters. Chapter 2 outlines the foundational concepts as well as related work on which this thesis is built. We start chapter 2 with an introduction to the DNS Infrastructure wherein we talk about concepts related to domain name space, resource records, name servers and resolvers. We also give a brief description of the DNS workflow and various kinds of attacks that can happen on DNS and thereafter we give a brief introduction to the DNSSEC protocol. The importance of formal methods is discussed in detail and the use of several variants of *modal logics* used for specification and verification of security protocols is also explained. Chapter 2 ends with the exploration of various validation tools available in the literature that can be used for validating security protocols. In chapter 3 we proposed a formal framework based on Modal logic called SVO logic that can be used to specify and verify the DNSSEC protocol. We proposed an extension of DNSSEC wherein two new axioms were added to the original SVO logic and by making use of the inference rules as well as axioms of SVO logic we were able to successfully derive proof for *data origin authentication* and *data integrity* of DNSSEC protocol. To the best of our knowledge we were the first ones to propose a logic based framework for analysing the DNSSEC protocol.

Contributions related to the development of validation tools for analysing the logical specifications of DNSSEC as developed in chapter 3 was carried out in chapter 4. We demonstrated how the DNSSEC protocol specification in Alice - Bob (A - B) notation can be verified using the AVISPA tool. Chapter 5 is a lengthy chapter in which we have two parts and in which we make two important contributions. The first part tries to address the analysis and verification of DNSSEC from a *Multi-Agent* perspective. In this part we extend a particular agent architecture called the BDI so as to incorporate temporal planning with explicit time-limits.

To this end we develop a temporal BDI architecture and demonstrate with the help of examples how temporal plan can be generated within a BDI architecture. In the second part we use this temporal BDI model for performance enhancement of the DNSSEC protocol. We conclude the thesis with chapter 6.

Chapter 2

Preliminaries, Foundational Concepts and Related Work

In this chapter we initially start with a discussion about the modern Domain Name System (DNS) wherein we talk about its current architecture as well as the reasons for having such a framework. Thereafter we outline the need for having a formal approach for analysing DNS and carry out a survey of the different logical frameworks proposed in the literature for reasoning about authentication protocols. We also outline in detail the validation tools available that could be used for the formal verification of authentication protocols.

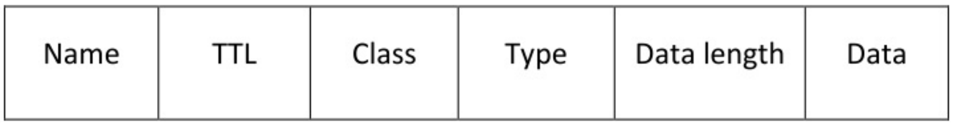
The main system used to translate human readable hostnames such as *uohyd.ac.in* to Internet protocol addresses like 183.82.173.128 is known as the Domain Name System (DNS). The DNS has a distributed hierarchical database structure and is the key behind every transaction on the internet as it is the primary system responsible for host-name translations. Modern Internet has its roots in ARPA Internet which was the first packet switched network in the world and was able to connect quite a few research centers in the United States during the seventies and eighties. As expected, managing a large network of many hosts becomes a challenge over a period of time as it becomes extremely difficult to identify the resource to which a user would like to connect to. Initially IP addresses helped in identifying the resources and later on host-names were created wherein a mapping was done between host-name and IP addresses which in turn was stored in a *hosts* file. The Network Information Center became a centralised repository for maintaining the complete `hosts file` list for the entire ARPA internet and as expected the centralised model became unmanageable. Though a distributed DNS having a tree hierarchy was proposed in 1983 to replace the centralised model the original DNS received a redesign in 1987 to become the base DNS being used today. The redesigned DNS was able to scale to the rapid growth of the Internet though it maintained the same concepts as the old one. Several variants of the Domain name System were proposed later on but overall it is still the same base system that performs the hostname to IP address translation.

2.1 DNS Infrastructure

In this section we explain the DNS architecture which includes the resource records, name servers and resolvers as well as the workflow of DNS.

2.1.1 The Domain name space and resource records

The domain name space [67] [66] basically describes how the resources (which includes all the devices that are connected to the Internet) are connected logically. In the domain name space, resources are connected in a hierarchical and logical tree structure. The domain name space was designed in such a way that it should be easy for scalability and for name-address resolution. Name of a domain is the list of labels on the path from a node to the root node of the tree. Resource information associated with a specific name is composed of separate Resource Records (RRs). Format of RR is as shown in Figure 2.1.



Name	TTL	Class	Type	Data length	Data

Figure 2.1: DNS Resource Record format

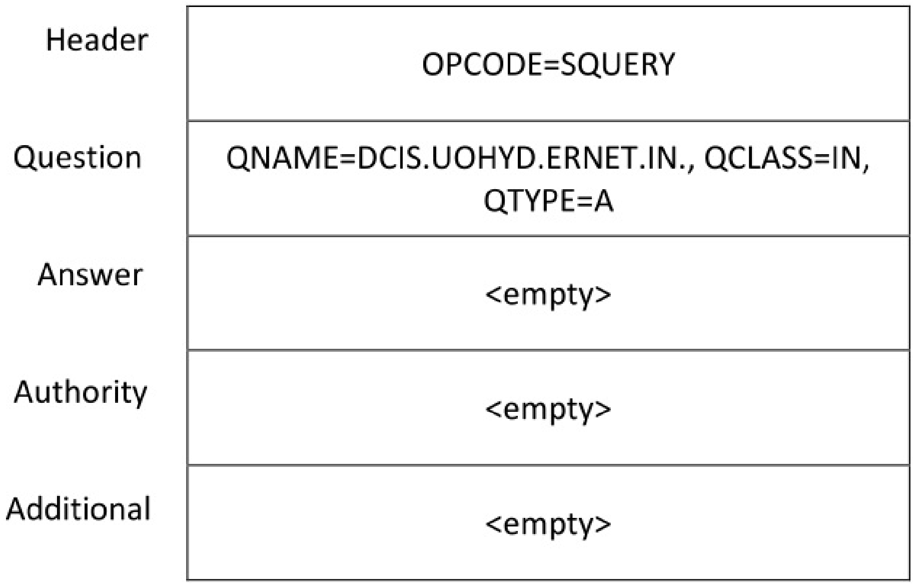
`Name:` indicates the name of a domain to which the record belongs to.
`TTL:` time to live indicates how long this record can be kept by the resolver in the cache memory for reuse.
`Class:` indicates which group the record belongs to.
`Type:` indicates the type of data the record holds. Different record types are as shown in the Table 2.1.
`Data length:` length of the data that is held in the record.
`Data:` the actual data carried by the record.

Type	**Description**
A	Internet Address
CNAME	Canonical Name (nickname pointer)
HINFO	Host Information
MX	Mail Exchanger
NS	Name Server
PTR	Pointer
SOA	Start Of Authority

Table 2.1: Types of Resource Records

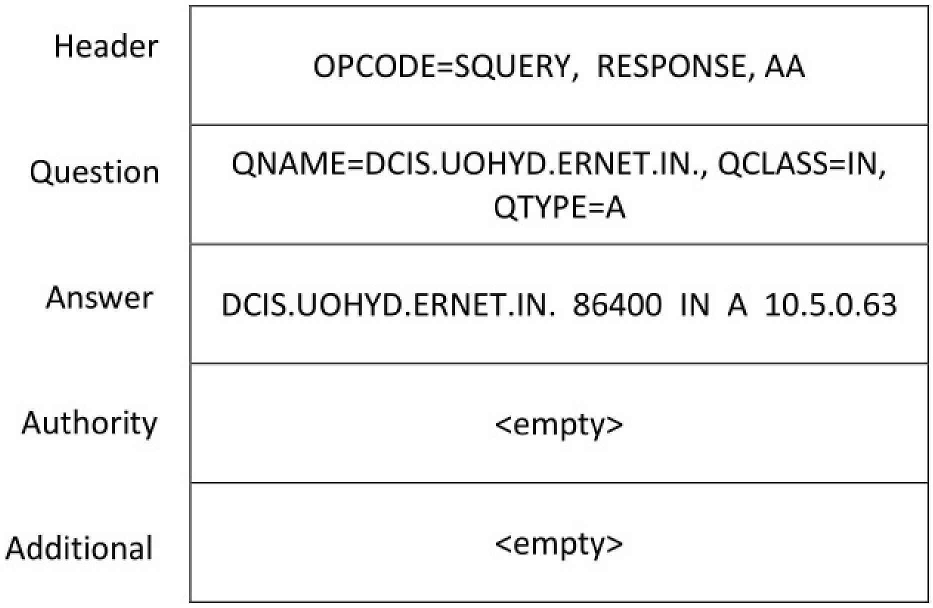
2.1.2 Name servers

The name servers are programs designed to hold the information about domain tree's structure. The server receives a query from the client and sends the response back. The typical format of the DNS query and the responses are as follows. The header of the query and response packets are similar, except that the RESPONSE bit is set in response packet, which indicates that it is a response. The Authoritative Answer (AA) bit is set if it comes from the actual server to which the domain belongs to. If the response comes from a different server which had cached the data, then AA bit will not be set. The question sections of both the query and the response are the same. The answer section of response indicates the required data for the given query. Authority section is used to send the *reference* of an authority server for a given query. Additional section is used to send extra and relevant information for a given query.



Header	OPCODE=SQUERY
Question	QNAME=DCIS.UOHYD.ERNET.IN., QCLASS=IN, QTYPE=A
Answer	<empty>
Authority	<empty>
Additional	<empty>

Figure 2.2: DNS Query format



Header	OPCODE=SQUERY, RESPONSE, AA
Question	QNAME=DCIS.UOHYD.ERNET.IN., QCLASS=IN, QTYPE=A
Answer	DCIS.UOHYD.ERNET.IN. 86400 IN A 10.5.0.63
Authority	<empty>
Additional	<empty>

Figure 2.3: DNS Response format

2.1.3 Resolvers

The resolvers are programs that contacts to name servers to retrieve information on behalf of the client requests. The resolvers are implemented using a set of library routines (e.g. libc) gethostbyname(char *name); gethostbyaddr(char *addr, int len, type); For example nslookup is an interactive resolver.

2.1.4 Work flow of DNS

A system that uses a distributed database deals with queries in either of the two ways: *recursive* or *iterative* manner. In recursive approach, first server pursues the query for the client with other servers until the query get resolved whereas in the iterative approach the server refers the client to another server and lets the client to pursue the query. Iterative servers can only answer information they know or have cached. Recursive name servers know how to ask others for information. Generally, local name servers are configured as *recursive servers* and top level name servers are configured as *iterative servers* [67, 66].

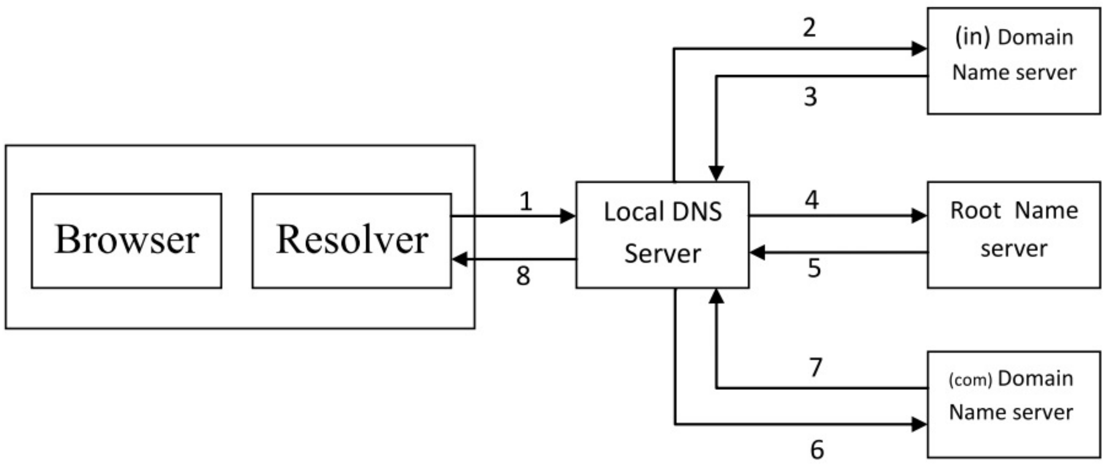


Figure 2.4: DNS query resolution in Recursive manner

Here we will illustrate the query resolving procedure with an example. We assume that the client at *uohyd* under *ernet* domain is sending a query to find an IP address of a system *D* under *com* domain. Please refer to Figure 3.1 for the visualisation.

1. Resolver at *uohyd* sends a query to local DNS server (at *ernet* domain): what is the IP address of *www.D.com*?

2. Local DNS server looks in its cache for IP address of www.D.com. If it is found, then it returns the same to the resolver. If it is not found in cache memory, then it forwards the query that has come from resolver to its parent domain name server (in our case it is *in* domain).

3. The name server at (.in) domain checks its database. If the answer to the query is found, then it returns the answer, else, it returns the *referral* (in our case it is *root* domain) to the local name server.

4. Local DNS server sends a query to the *root* domain name server.

5. The name server at *root* checks its database and returns the *referral* (in our case, it is *com* domain) to the local name server.

6. Local DNS server sends a query to *com* domain name server.

7. The name server at *com* domain checks its database and returns the IP address of D.com if it is found, else, it returns 'no such name is found under *com* domain' to local name server.

8. Local DNS server forwards the response back to the resolver.

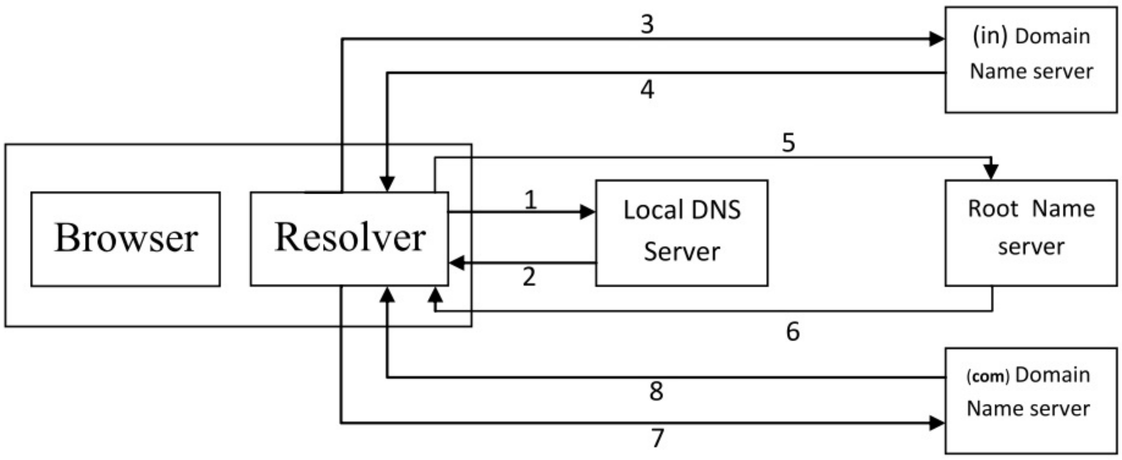


Figure 2.5: DNS query resolution in Iterative manner

1. Resolver sends a query to local DNS server (at *ernet*: what is the IP address of *www.D.com*?)

2. Local DNS server looks in its cache for IP address of www.D.com, if it is found, then it returns the same to the resolver. If it is not found in cache memory, then it returns the *referral* (in our case it is *in* domain) to the resolver.

3. The resolver sends a query to the *in* domain name server.

4. The name server at (.in) domain checks its database. If answer to the query is found, then it returns the answer, else, it returns the *referral* (in our case it is *root* domain) to the resolver.

5. The resolver sends a query to the *root* domain name server.

6. The name server at *root* checks its database and returns the *referral* (in our case it is *com* domain) to the resolver.

7. The resolver sends a query to the *com* domain name server.

8. The name server at *com* domain checks its database and returns the IP address of D.com if it is found, else, 'it returns no such name is found under *com* domain' to the resolver.

After understanding the working procedure of DNS system, we now explore different types of attacks on the DNS system.

2.2 Attacks on DNS

At the time when DNS was invented, the need for the security of the system was not thought of. But as Internet evolved and many critical services were deployed, the security of the DNS system has become more important. Given below are some of the most prominent attacks that has taken place on the DNS system [7].

2.2.1 Attacks on DNS Protocols

1. DNS cache poisoning: maliciously modifying the data stored at DNS servers, aiming at either redirecting the traffic to a malicious website with the intention to lure the sensitive data such as pin numbers, passwords, etc or to initiate Denial Of Service (DOS) attack.

2. DNS spoofing attacks: creation of IP packets with a forged source IP address aiming at creating DOS attack, unauthorised access, etc.

3. Man In The Middle (MITM) attack: where a malicious intruder impersonates one or both parties to gain the access to the information that the two parties were exchanging.

4. Phishing attacks: aiming to redirect a website's traffic to another bogus website. Normally the hackers make use of *Distributed Denial of Service (DDoS)* as a prime tool to suspend Internet services for the legitimate users.

2.2.2 Attacks on DNS in Real time Scenario

Some of the major attacks on the DNS system in real time scenario are given below. The first three attacks happened on root DNS servers and the other attacks happened on normal DNS servers.

1. November 30, 2015 and December 1, 2015, many root name servers received approximately upto 5 million query requests per second, for a single anonymous domain name.

2. February 6, 2007, an attack due to which minimum two of the root servers got affected severely while the other two experienced heavy traffic for 24 hours.

3. October 21, 2002, an attack on all 13 DNS root servers by sending huge number of ICMP (Internet Control Message Protocol) packets using *boot net*

had occurred. The consequence of this attack was that for approximately one hour there was no Internet connectivity.

4. October 21, 2016, the Dyn cyber attack which is a kind of multiple Denial-Of-Service attack due to which huge number of legitimate users from Europe and North America could not access the Internet platforms and services.

5. September 20, 2016, when the record for the largest DDoS attack was broken. This attack was on Akamai, topping the charts at 665 Gbps. This DDoS attack is the largest attack that has ever been witnessed till date.

2.3 DNSSEC

DNSSEC was designed in the mid 1990's mainly because of the dangers of DNS cache poisoning [18] and was meant to authenticate and protect the integrity of DNS through the use of public key-based digital signatures. The current version of DNSSec came out in 2005 after the initial trial versions [40, 41] and it has been shown that it is able to defeat cache poisoning attacks as the authenticity and integrity of the *responses* can be cryptographically verified. Unless and until the underlying cryptography is broken the *responses* cannot be forged and the resolvers cache would remain uncorrupted as the resolvers will only forward or cache authentic responses.

As mentioned earlier DNS has a hierarchical structure and DNSSEC [74, 30, 46, 51, 43, 17] leverages this `dot` separated hierarchy so as to delegate trust from a parent to the child entity. To consider an example, the public key (DNSKEY) for google.com is signed by .com and the DNSKEY for .com is signed by the root servers. The public key of the root server is to be distributed to each resolver with the help of a DNS resolver software. Hierarchical trust delegation within a zone is also allowed in DNSSEC in which a zone can have many signing keys (DNSKEY records) like Zone Signing Keys (ZSK) as well as Key Signing Keys (KSK). A strict hierarchical trust model is followed by DNSSEC with multiple rooted trees as can be evidenced from the fact that a parent entity holding a DS Record (explained in the next chapter) can sign in many KSKs and this can lead to signing in of many ZSKs which in turn can lead to the signing in of many records. It is a strict policy in DNSSEC that across distinct domains trust cannot be delegated. It is possible that the same name may be owned by an entity under different top level domain wherein one name points to the other with a cross domain name as in google.com and google.ca. In order to provide secure communications DNSSEC provides two characteristic features like *integrity* and *authentication.* The use of cryptographic signatures for each resource record provides integrity whereas construction of *chain of trust* from a third party parent we *believe* in and the belief that the process with which the third party verifies that the DNS updates are from the correct entity provides authentication. We explain in detail about the functioning of DNSSEC in the next chapter. In the next Section we describe integrity and authentication based on their types as well as the different methods available to verify them.

2.4 Integrity

Integrity of data can be defined as an assurance that the received data is the same as that is sent (i.e., contains no modification, insertion, deletion, or replay) and in a way assures the trustworthiness of the data. Trustworthiness of data can be also adjudged by data origin authentication which means that whatever the sender is sending should be received by the receiver without any changes and the receiver must verify the genuiness of the originator of the data received. Thus data integrity has several dimensions as discussed below.

2.4.1 Types of Data Integrity

Data integrity can be classified into two types they are — Physical data integrity, Logical data integrity.

1. **Physical Data integrity:** Physical data integrity deals with storing, retrieving and protecting data from internal and external events. The internal events may include electro mechanical faults, design flaws, material fatigue,

corrosion etc. The external events may include power failures, natural disasters, wars and terrorism, and also other kind of special environmental hazards like ionizing radiation, extreme temperatures, pressures etc. Physical data integrity often make use of techniques like *checksum, parity encoding, error detection, Random Array of Inexpensive Disk (RAID)* etc. to check physical data integrity.

2. **Logical Data integrity:** Logical data integrity is related to the correctness or rationality of data in a given context. It also includes concepts like referential integrity and entity integrity in a relational database. Logical data integrity problems may include software bugs, design flaws and human errors. Common methods employed to ensure logical integrity include – checking the constraints, foreign key constraints, program assertions etc.

2.4.2 Causes of Integrity Violations

Violations of integrity may occur due to hardware or software malicious activities, malfunctions or inadvertent user errors.

- **Hardware and Software Errors**

  Data sent across the network or stored on storage device, may get modified/ corrupted because of hardware or software malfunctioning. Proper protocols like IPsec, Transport Layer Security(TLS), DNSSEC etc., must be designed to protect data from unauthorized modifications.

- **Malicious Intrusions**

  Trustworthy data management in a computer system is a very vital challenge that hardware and software designers face today. In order to avoid intrusions particularly, to the critical and vital data sources, sophisticated authentication mechanisms must be implemented and also proper access control mechanisms must be implemented.

- **Inadvertent User Errors**

  Errors committed by the user at application level may sometimes become the causes for data integrity violation. An accidental deletion of a file from a database may result in DBMS not functioning in the desire manner as well as leads to data corruption. In general, violations of implicit assumptions of the applications by the user actions are the root cause for data integrity violations.

2.4.3 Methods to Verify Data Integrity

The following techniques use cryptographic hash functions and encryption techniques to achieve data integrity.

1. **Message Authentication Code (MAC)**

   Message Authentication Code (MAC), also known as a `keyed hash function` is one of the cryptographic technique to achieve data integrity. MACs can be used to ensure integrity of the data exchanged between two parties that share a secret key. This function takes a key and data as input and produces hash value, which is known as MAC value as output. Sender appends the MAC value to the data and sent to the receiver. The receiver computes new MAC value with the received data and compares it with the received MAC value. If both are matched then the receiver can conclude that data did not get modified during transit and thereby data integrity is achieved. An attacker who changes the data cannot modify the hash value without knowing the value of the secret key.

   Message Authentication Code (MAC) can also be generated by using hash functions and encryption techniques. In this method it is assumed that two communicating entities who want to ensure data integrity during data exchange are sharing secret keys (either symmetric keys or asymmetric keys). When data confidentiality is not required, then only the output of hash function (message digest) is encrypted with symmetric key which is shared by both communicating entities. The encrypted message digest is called MAC. The MAC is appended to the message and transmitted to the other side. The other side MAC value is first decrypted and compared with the newly generated hash value from the received message. If both the values are matched, then it is ensured that the data is not modified in transit and data integrity is achieved.

2. **Digital Signature**

   This method uses public key cryptography. The sender encrypts the output of the hash function with the private key of the sender, which we call it as Digital Signature (DS). This signature is appended to the message. The receiver decrypts the signature using public key of the sender and compares with newly generated hash value from the received message. If both values are matched, then it is ensured that the data is not modified in-between and thus data integrity is achieved.

3. **Shared Secret Value**

   This method does not use encryption techniques and uses only hash functions to achieve data integrity. In this technique, the sender will append the secret value to the message and applies the hash function function to generate the message digest. This message digest is appended to the message and transmitted to the receiver. The receiver appends the secret value to the received message and applies the hash function. If the received hash value and generated hash value is matched, then it ensures that the message is not modified in-between and data integrity is achieved.

2.5 Authentication

Authentication is a security mechanism that provides an assurance that the communicating entity is the one that it claims to be. This in a way means automatic verification of the identity of a person or a system or an entity who tries to access the system or resource. The main purpose of authentication is to provide confidence that an entity is not performing either a *masquerade* or an *unauthorized replay* of a previous connection. It also confirms that the data has originated only from the intended source and thus can lead to *message Authentication* or *data origin authentication* and *non repudiation.*

2.5.1 Types of Authentication

Peer Entity Authentication is one type of authentication mechanism used to provide confidence in the identity of the entities which are logically connected for communication. *Data-Origin Authentication* provides assurance that the source of the received data is the same as it is claimed.

2.5.2 Factors used in Authentication

In order to authenticate or to assure the identity of an entity we use any one of the three or combination of two or more of the following factors.

1. Knowledge factors: Based on user's knowledge like passwords, Personal Identification Number (PIN), security questions etc.

2. Ownership factors: Based on user's possession like ID card, cell phone holding a message or a software token, wrist band etc.

3. Inherence factors: Based on what the user *is* or *does* like signature, face, voice, fingerprint, DNA sequence, retinal pattern etc.

2.5.2.1 Types of Authentication based on Number of Factors used

This classification is based on the number of factors used to verify the identity of an entity.

1. **Single-factor authentication:** As the name implies, only one type of factor out of the three is used to verify the claimed identity. This kind of authentication is applied only in limited applications where possibility of security attacks is less and also the consequences of security breach does not have much impact on the resources that is accessed. It is the weakest level of authentication. This kind of authentication is not suggested for finance

related tasks. Usually they require a higher level of security. *Eg:* Logging into personal systems.

2. **Two-factor authentication (2FA):** Two-factor authentication is a technique used to confirm the identity of an user by making use of two components belonging to two different factors. For example if we need to withdraw money from the ATM machine user should possess a valid bank card as well as know the correct PIN.

3. **Multi-factor authentication (MFA):** It is similar to 2FA, but it can combine more than 2 authentication factors to extend the level of security. Multiple authentication factors are used to achieve high level of security.

4. **Strong authentication:** In order to achieve strong authentication, the individual elements that are chosen as factors must be *mutually independent* (that means, no relation between factors so that even if one factor gets compromised other one will be safe) and at least one must be *non-reusable* and *non-replicable* (except for inherence factor) with respect to usage or time. For instance, consider the use of one-time-password which is only valid during a short period of time. Since 2013, strong authentication has been in use within the European Union and the SEPA payment zone for remote payment transactions. Strong authentication is particularly used where access to an account must be linked to an actual person, corporation or trust. The Reserve Bank of India (RBI) has also followed the European Central Bank (ECB) lead and mandated strong authentication for online transactions.

2.5.3 Methods to Verify Authentication

1. **Message Authentication Code (MAC)** and **Digital Signature** has been already explained in the Section 2.4.3 which is related to methods to verify data integrity.

2. **Authenticated Encryption (AE) or Authenticated Encryption with Associated Data (AEAD):**
   AE is one type of encryption technique which can be used to provide confidentiality, integrity, and authentication of the data simultaneously. Approaches to Authenticated Encryption are as follows: (1) Encrypt-then-MAC (EtM): In this technique first the plain-text is encrypted and thereafter the MAC value is computed on the resulting cipher-text. The cipher-text and its corresponding MAC are sent together. This technique is applied in the Internet Protocol Security (IPSec). (2) Encrypt-and-MAC (E & M): In this technique, first a MAC value is computed on the plain-text, then the plain-text alone is encrypted to generate cipher-text. Now the MAC value and the cipher-text are sent together. This technique is applied in Secure Shell (SSH) (3) MAC-then-Encrypt (MtE): In this technique first a MAC value is computed on the plain-text and then the cipher-text is produced based on a combination of both plaint-ext and MAC value. This technique is used in Transport Layer Security (TLS/SSL).

In the next section we outline the importance of formal methods for analysing security protocols and give an overview of the different existing methods.

2.6 Formal Methods for Security Protocols

Formal methods have been applied to cryptographic protocols to analyze the distribution of keys between communicating principals. Distribution of keys could be done either by making use of a key server or generation of keys by themselves. Although the problem looks simple, when it comes to reality it is not so, an this is because of the hostile intruder, who can always disturb the system. Whenever any cryptographic protocol is designed, the intruder who can intercept, replay, delay, delete or modify the messages and also sometimes maintains league with some dishonest principals, should be taken into consideration. In present day computer networks, Internet services plays a vital role. In order to provide critical service like financial transactions in a smooth and fair manner, it is mandatory to use robust cryptographic protocols. The job of cryptographic protocols is becoming complex because apart from providing the intended services, they also should have some mechanism to deal with the intruders, who may attempt to learn secrets or impersonate honest principals, perform denial of service attacks, traffic analysis etc. Therefore for any cryptographic protocol it is a big challenge to achieve its goals in the face of hostile intruders. Basically the attacks that the intruders perform are nonintuitive in nature and therefore it is very difficult to capture them. Security protocols are inherently very complex and generally implement one or more cryptographic techniques to ensure that the various specified security goals such as confidentiality, authenticity, integrity, availability, anonymity, fairness, etc. are maintained. Even though the security protocols are designed and implemented with utmost care, there are a large number of instances in which it is proved that many of the security protocols which have been in use for many years have *logical flaws* hidden inside. For instance, though the Needham-Schroeder protocol [69] looks like a very simple protocol to analyse, a logical flaw [54] found in it only after 17 years of it use. The above mentioned protocol clearly shows how much hard it is to analyze and discover security threats even with a simple protocol. It is also proved that the secrecy preservations are *co-NP-complete* for a bounded number of sessions [75], and decidable for an unbounded number of sessions under some additional restrictions [4, 19, 78].

2.6.1 Classification of Formal Methods

Formal methods are broadly classified into three types.

1. **Methods Based on State Machines or Attack Construction Method** In this method, the participants of a protocol are modeled as communicating *state machines* [29]. It is assumed that the messages exchanged by the participants are passed through the intruder, who is capable of modifyying, destroying, delaying or doing nothing. In attack-construction methods vulnerabilities are discovered by using *algebraic properties* of protocol's algo-

rithms. Most of the attack-construction techniques explore the state space to discover a path to an insecure state. These methods are applied for ensuring authentication, correctness and other security properties. The main drawback of these methods is that they have to analyse very large number of possible events before concluding the remarks.

2. **Methods Based on Modal Logic or Inference Construction Method** In this method the messages exchanged in a distributed system are represented as a set of beliefs or knowledge in the framework of Modal logic. Inference rules are applied to derive beliefs from other beliefs and/or knowledge. In this method protocol analysis start with an initial set of beliefs that are assumed to be true. Each new message that is received is considered as a new set of beliefs. Finally the inference rules are applied to derive the security parameters of a given security protocol from the set of initial and gained/updated/revised beliefs. If the goals of the security protocol are derived successfully, then we can conclude that the protocol is correct, otherwise the protocol could be suspected with *security flaws*. The most widely used modal logics to verify the security protocols are `BAN logic` [24] , `GNY logic` [45], Higher Order Logic `(HOL)` and `SvO logic` [79],

3. **Methods Based on Algebras to Reason About Knowledge or Proof Construction Method** Proof-construction methods first model the computations that are performed by the protocol and then the security properties are defined as theorems. Automated theorem provers are used to verify these theorems and thereby prove properties of the model. Once the security properties are specified as theorems, four theorem proving techniques are used to verify the security properties of a given security protocol. (i) **Proof by construction or direct proof:** This method is the simplest and easiest one for proving a given theorem. This method is a two step process as shown below. For example to show p $\rightarrow$ q, we start with the assumption that P is *true* and

now use P to show that 'Q' must be true when P is true. (ii) **Proof by contradiction:** This method works based on the fact that any proposition must be either true or false, but not both simultaneously. To prove the statement, we start with assumption either true or false and we arrive at contradiction to the assumption, which shows that the statement is *true* and *false* simultaneously, which violate basic rule, hence the proof. For example, by assuming both P and $\urcorner$ Q are simultaneously true and deriving a contradiction we can show that P $\rightarrow$ Q. (iii) **Proof by induction:** Proof by induction method uses recursion to show an infinite number of facts in a finite amount of space. In this method, the proof is shown in three steps. In the first step it is shown that a propositional formula P(x) is true for some base case. In the second step, assume that P(n) is *true* for some n, and show that this implies that P(n + 1) is *true.* In the third step, show by the principle of induction that the propositional formula P(x) is *true* for all n greater than or equal to the base case. (iv)**Proof by contraposition:** Proof by contraposition method is based on the fact that we know that the implication P $\rightarrow$ Q is equivalent to $\urcorner$ Q $\rightarrow$ $\urcorner$ P. The second proposition is called the contrapositive of the first proposition. By saying that the two propositions are equivalent, we mean that if one can prove P $\rightarrow$ Q then they have also proved $\urcorner$ Q $\rightarrow$ $\urcorner$ P , and vice versa. Proof by contraposition is a more effective approach when a direct proof method becomes complex and tricky.

In the next section we discuss in detail the different modal logics used for the formal specification and verification of security protocols.

2.7 Modal Logics to Verify Security Protocols

Modal logic extends classical logics like *propositional logic* and *predicate logic* to include *modalities* such that these modalities when prefixed to a declarative sentence can be understood to modify the way in which it is true. For instance, with the help of modalities one should be able to say that a statement/proposition could be `necessarily/possibly` true or true in the `future/past` or is `known/believed` to be true or it is true `after` performing a certain computational operation α etc. This kind of representation and reasoning is not possible using propositional/predicate logic. There are different types of modalities. Modality used to express different kinds of *truth* are called *alethic modalities*, the modalities used to express *time* are called *temporal modalities*, the modalities used to express *obligations* are called *deontic modalities*, the modalities used to express *knowledge* are called *epistemic modalities* and the modalities used to express *belief* are called *doxastic modalities.* The modalities used to analyse security protocols are *epistemic* and *doxastic.* Modal logic makes use of two operators denoted by $\Box$ and $\Diamond$ interchangeably to represent and reason about the different types of modalities as mentioned above. We can represent the two operators as follows: If φ is a proposition then

$$\Box\varphi = \neg\Diamond\neg\varphi \quad \Diamond\varphi = \neg\Box\neg\varphi$$

Differently from negation or implication, this operator is not intended to be truth-functional, i.e., its meaning does not depend only on the truth-values of the sub-formulae. The intended meaning of $\Box\varphi$ is to qualify the truth value of φ: If φ is true then, $\Box\varphi$ specifies that φ is not only true but necessarily true, i.e., is true independently from the scenario (or state, world, etc.). It is also possible to use modal logics to interpret a formula like $\Box\varphi$ as φ is known by the agent a, φ is part of the knowledge of a, and φ is believed by the agent a etc. It is also possible to express properties which cahracterizes a modal operator by means of a set of axioms as follows: Suppose for example consider the modal operator $\Box$, then $\Box\varphi \to \varphi$ can express the fact that everything that is necessarily true is also true and aslo that what is known by the agent a must be true. In the next subsections various modal logics like BAN, GNY, AT and SVO are discussed.

2.7.1 Burrows Abadi Needham (BAN) Logic

BAN logic [24], was developed by Burrows, Abadi and Needham and was the first modal logic used to reason about security protocols. BAN is based on a set of beliefs held by two honest/truthful principals/parties. It works under the principle that *if two honest principals correctly execute a protocol then the beliefs that result out of that protocol execution can itself be used to analyse the authentication/security protocol.* For example suppose that we want to prove that Alice *believes* in a key that was received from the server as a good key to communicate with Bob for the current session. Then the approach followed by BAN to derive the goals of a given protocol is as follows: Firstly, the messages in the protocol specification are idealised into logical formulae, i.e., if Alice receives a session key from the server in the form of an encrypted message, then the key may be replaced by a formula which in turn means that the key is good. Now, based on Alice's capabilities to decipher the key and other assumptions that may hold, it might lead to the conclusion that Alice is *believing* the key that it received from the server and also that it is a good key to communicate with Bob.

2.7.1.1 BAN Notation

In order to keep simple and more intuitive the original BAN notations are replaced with almost plain English formulas. The language of BAN consists of the following expressions:

- `P believes X` : Principal P knows and acts as if X is true.
- `P received X` : Principal P has received a message that contains data X.
- `P said X` : Principal P has sent a message containing X at some prior point, and P believed X and knows that he/ she was sending X at that time.
- `P controls X` : Principal P has jurisdiction on X.
- `fresh(X)` : X has not been sent in any message prior to the current protocol run.

- $\mathtt{P} \xleftrightarrow{k} \mathtt{Q}$: Principals P and Q believed that k is a good key to communicate with each other and also trusted that k will never be discovered by any principals other than P, Q and any principal trusted by P or Q.

- $\mathtt{PK(P,k)}$: k is a public key of principal P and corresponding private key k^{-1} will never be discovered by any principal other than P.

- $\{\mathtt{X}\}_k$: X is encrypted with k from principal P.

2.7.1.2 BAN Rules

The following BAN logic rules are used in deriving the conclusions during analysis of protocols. Among all *message meaning* and *nonce verification* rules are the central rules of BAN logic.

1. **Message Meaning**

$$\frac{P\ believes\ P \xleftrightarrow{k} Q\ \wedge\ P\ received\ \{X\}_k}{P\ believes\ Q\ said\ X}$$

If principal P receives X encrypted with k and if P believes k is a good key for talking with principal Q, then P believes Q once said X.

- **Public key version of message meaning**

$$\frac{P\ believes\ PK(Q,\ k)\ \wedge\ P\ received\ \{X\}_{k^{-1}}}{P\ believes\ Q\ said\ X}$$

If principal P receives X encrypted with private key k^{-1} of principal Q and if P believes k is the public key of Q, then P believes Q once said X.

2. **Nonce Verification**

$$\frac{P\ believes\ fresh(X)\ \wedge\ P\ believes\ Q\ said\ X}{P\ believes\ Q\ believes\ X}$$

This rule allows promotion from the past to the present. In order to do so one condition that should be satisfied that is, X should not contain any encrypted text.

If principal P believes that X is fresh and P believes that principal Q said X once then P believes that Q believes X.

3. **Jurisdiction**

$$\frac{P\ believes\ Q\ controls\ X\ \wedge\ P\ believes\ Q\ believes\ X}{P\ believes\ X}$$

If principal P believes principal Q controls value X and P believes Q believes X then P believes X.

4. **Belief Conjuncatenation**

$$\frac{P\ believes\ X\ \wedge\ P\ believes\ Y}{P\ believes\ (X, Y)}$$

$$\frac{P\ believes\ Q\ believes\ (X, Y)}{P\ believes\ Q\ believes\ X} \qquad \frac{P\ believes\ Q\ said\ (X, Y)}{P\ believes\ Q\ said\ X}$$

The above rules are related to concatenations of messages/conjunctions of formulas. Concatenations of messages and conjunctions of formulas are not distinguished in the above rules both are represented as (X, Y). If principal P believes X and P believes Y then P believes concatenation/ conjunction of X and Y. If principal P believes principal Q believes concatenation/ conjunction of X and Y then P believes Q believes X. If principal P believes principal Q said concatenation/ conjunction of X and Y then P believes Q said X.

5. **Freshness Conjuncatenation**

$$\frac{P\ believes\ fresh(X)}{P\ believes\ fresh(X, Y)}$$

If principal P believes X is fresh, then P believes any message containing X is also fresh.

6. **Receiving Rules: Seeing is Receiving**

$$\frac{P\ believes\ P \overset{k}{\longleftrightarrow} Q \ \wedge\ P\ received\ \{X\}_k}{P\ received\ X} \qquad \frac{P\ received\ (X, Y)}{P\ received\ X}$$

   A principal receiving a message also receives submessages he can uncover. BAN logic does not distinguish from what a principal possesses from what he/ she has received in some message. If principal P believes k is a good key to communicate with principal Q and P received the message X encrypted with key k then we can say P has received X. If principal P receives concatenation/ conjunction of X and Y then we can say that P has received X.

2.7.1.3 Analysis of Protocol using BAN Logic

Given the above rules there are four steps to be followed to analyze a protocol using BAN logic: 1) Idealization of a given protocol 2) Stating the initial assumptions 3) Annotating the protocol: For each message transmission "P $\rightarrow$ Q : M" in the protocol, assert Q received M 4) Applying BAN rules on the assumptions and assertions to derive beliefs held by other principals (In other words using the logic to derive the beliefs held by the protocol principals). We illustrate the analysis of a protocol using BAN logic by taking a simple protocol like the *Wide-mouthed-frog protocol.* The prime requirement in the analysis of any protocol using formal logics is thorough the understanding of the protocol in terms of its working procedures, assumptions etc.

Wide-mouthed-frog Protocol

It is a very simple protocol consisting of two principals (A, B) and a server (S). It transfers a key from A to B via S in only two messages by using synchronized clocks, and by allowing A to select the session key.

$$Message\ 1\ A\ \rightarrow\ S : A,\ \{T_A, B, K_{AB}\}_{K_{AS}}$$
$$Message\ 2\ S\ \rightarrow\ B : \{T_S, A, K_{AB}\}_{K_{BS}}$$

The pictorial representation of the Wide mouthed frog protocol is shown in Figure 2.6.

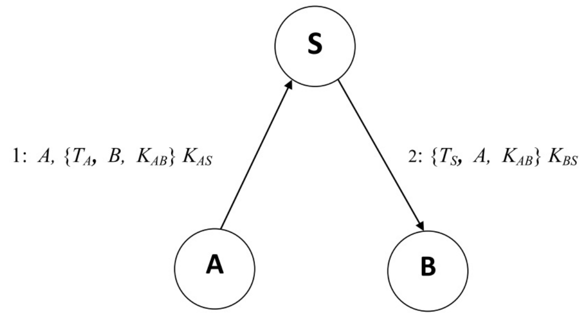


Figure 2.6: Wide mouthed frog protocol

Wide mouthed frog protocol analysis using BAN logic is as follows:

Step 1: Protocol idealization is as shown below.

$$\begin{aligned}&Message\ 1\ A\ \rightarrow\ S : \{T_A, (A\ \overset{K_{AB}}{\longleftrightarrow}\ B)\}_{K_{AS}}\\ &Message\ 2\ S\ \rightarrow\ B : \{T_S, A\ believes\ (A\ \overset{K_{AB}}{\longleftrightarrow}\ B)\}_{K_{BS}}\end{aligned}$$

The main intuition behind the idealisation step is not only to have an understanding of the working of the protocol but also to announce the beliefs that the principal would be having at the time of sending the messages. It is very clear from the idealisation given above for message 1 wherein B, K_{AB} is replaced with $(A\ \overset{K_{AB}}{\longleftrightarrow}\ B)$ (A's beleif about K_{AB}) to indicate that K_{AB} is a good key for communication between A and B. In the same manner second message attaches S's belief about K_{AB}. The formula $(A\ \overset{K_{AB}}{\longleftrightarrow}\ B)$ in message 1 and 2 also indicates A and S believes K_{AB}.

Step 2: Initial assumptions about the protocol are as shown below.

$$\begin{array}{ll} A\ believes\ A\ \overset{K_{AS}}{\longleftrightarrow}\ S & B\ believes\ B\ \overset{K_{BS}}{\longleftrightarrow}\ S\\ S\ believes\ A\ \overset{K_{AS}}{\longleftrightarrow}\ S & S\ believes\ B\ \overset{K_{BS}}{\longleftrightarrow}\ S\\ S\ believes\ fresh(T_A) & B\ believes\ fresh(T_S)\\ \multicolumn{2}{l}{B\ believes\ A\ controls\ A\ \overset{K}{\longleftrightarrow}\ B}\\ \multicolumn{2}{l}{B\ believes(S\ controls(A\ believes\ A\ \overset{K}{\longleftrightarrow}\ B))}\end{array}$$

The most suspicious assumptions are that A knows the session key K_{AB} in advance, and B trusts A in the generation of session key (B *believes* A *controls* $A \overset{K}{\longleftrightarrow} B$).

Step 3: Following are the assertions that can be made using step 1.

$$S \textit{ received } A,\ \{T_A, B, K_{AB}\}_{K_{AS}}$$
$$B \textit{ received } \{T_S, A, K_{AB}\}_{K_{BS}}$$

Step 4: The analysis is almost trivial.

Since the protocol execution starts with A by generating the key K_{AB} and thereafter sends it to server(S), A has jurisdiction over its freshness and intended use. Therefore we can write:

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

S receives the first message from A and interprets it, therefore we can write:

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

S then sends the message to B, where it receives and interprets. From the rules of message meaning, nonce verification, and jurisdiction, we can obtain the following beliefs:

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

The most dubious assumption in this protocol is that B trusts A to generate and control good keys that can be used to protect the communication between A and B. Normally simple principals are not allowed to generate and control the session keys because they could simply reveal the keys to third party. For this reason jurisdiction over generation of session keys would be given to only *trusted servers.* Therefore in order to use this protocol practically principal A has to be tread as trusted source instead of simple principal participating in protocol execution. The process of protocol analysis using BAN logic is shown in Figure 2.7.

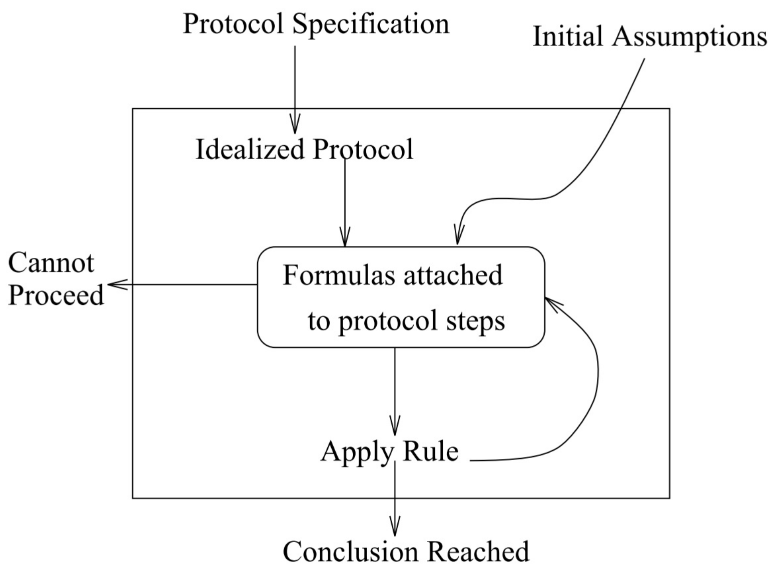


Figure 2.7: Protocol analysis with BAN logic

BAN logic was able to successfully find flaws in some of the well known protocols like Needham-Schroeder protocol, Andrew secure RPC handshake protocol and CCITT X.509 protocol. BAN logic also uncovered redundancy in Needham-Schroeder protocol, Otway-Rees protocol, Kerberos, Yahaloom protocol, Andrew RPC handshake protocol and CCITT X.509 protocol. In 1990, Nessett demonstrated a significant flaw in BAN logic [70]. In order to show imperfections in BAN logic Wessels [88] applied it to verify *Station to Station protocol* and concluded that BAN logic cannot handle *multi-role* attack. Other limitations of BAN logic are, it heavily depends on the initial assumptions and protocol idealization. The idealisation is difficult to prove correct, but in BAN logic idealization is one of the most critical step in analysing the protocol. BAN logic cannot handle man-in-the-middle attack and explicit arithmetic in protocols. BAN logic also does not differentiate between a principal *seeing* a message from *understanding* it.

2.7.2 Gong Needham Yahalom (GNY) Logic

GNY logic [45] was developed by *Gong, Needham, and Yahalom* with the intention of more scope than it's well known predecessor called *BAN logic*. The main aim behind the development of *GNY logic* is to overcome the limitations of *BAN logic*, to keep the analysis procedure simple, at the same time provides an option to the user to specify the required assumptions explicitly and deriving the conclusions. In order to improve the consistency in analysis and to introduce new capability of reasoning at more than one level, *GNY logic* takes new approach where *content* and *meaning* of messages are treated separately. GNY logic makes the distinction between possessing a message and believing a message. In GNY logic, it is possible to express the ability of principals to anticipate the near future messages that they are going to receive, this property is known as *recognizability.* GNY logic could distinguish certain messages that are not replays of a receiver's own previous messages in a given session. The main drawback of GNY logic is it is very complex and it has more than 40 inference rules that are to be consider at each step compare to other logics.

2.7.3 Abadi Tuttle (AT) Logic

AT logic [3] was proposed by Abadi and Tuttle. AT logic reformulates BAN logic and provides a new model theoretic semantics. It introduces new notions like "P actually possess the key K" denoted as "P has K" and "P has sent X" in the present denoted as "P says X", to provide more direct definitions to dispense with an implicit assumption of honesty. AT logic improves syntax and inference rules of BAN logic by doing some modifications. In the first modification, AT logic treated arbitrary expressions and formulas differently by defining the language of messages. Second, it rewrote the inference rules as axioms by providing all the propositional connectives such as negation, disjunction and implication. Third, all concepts are defined independently and formulated as axioms by simplifying inference rules and leaves with only modusponens and necessitation rules. This allows an analyzer to assess the truth of assumptions rigorously. AT logic provides a detailed computation model and is very expressive. AT logic provides a formal syntax and semantics and unlike BAN, GNY, VO; AT logic is more close to traditional modal logic. The disadvantages of AT logic is that it assumes perfect encryption and it does not address idealization step of the BAN logic.

2.7.4 Van Oorschot (VO) Logic

VO logic [85] was developed by Van Oorschot. The main objective of VO logic is to extend BAN family of logics in a manner that allows authenticated key agreement protocols to be analyzed, and in a better way to examine the goals and beliefs in the protocols. In order to fulfil the objective it refined the BAN logic construct ”shares the good crypto key”. It also defined a new key for confirmation primitive and new postulates for reasoning about ”jointly established” keys. By making use of the extensions it was able to analyze a new set of protocols and allowed for a closer analysis of the goals and beliefs of the new set of protocols. Disadvantage of VO logic is that the time and message ordering is not addressed.

2.7.5 SyVerson-van Oorschot (SVO) Logic

SVO logic [79], was developed by Syverson and van Oorschot. The main goal behind the development of SVO logic is to combine the best features available in the four logics namely, `BAN, GNY, AT, and vO` into a single logic. Apart from that, the *SVO logic* provides its own model-theoretic semantics with respect to which the logic is sound. We can use *SvO logic* to express and/or specify any security protocol with minimum effort. The SVO logic can be used to provide reasoning about a security protocol. The designers have taken utmost care to keep it’s usage simple and at the same time with more expressiveness. In the next Chapter we make use of the *SVO logic* for formal reasoning about DNSSEC protocol.

2.8 Automation Tools to Validate Security Protocols

The first work that was done in the area of automatic tools for validating security protocols is by Dolev and Yao [38] and later by Dolev et al. [39]. They developed a set of polynomial-time algorithms for deciding the security of a restricted class of protocols. But the problem with this model is that even a slight relaxation of the restrictions on the protocols would make the security problem undecidable. The later work concentrated on developing tools to analyze all types of security protocols in general. Almost all the tools are based on the Dolev-Yao model or some variant. These tools include Interrogator [65], AVISPA [9] [80], Naval Research Laboratory (NRL) Protocol Analyzer [58], and the LongleyRigby tool [52]. Most of these tools were successful in finding flaws in protocols which were undetected by human analysts. In the following subsections we discuss various automation tools in the literature to validate security protocols.

2.8.1 NRL Protocol Analyzer

NRL protocol analyzer [58] models the protocol as a set of state machines which can interact with each other. It then tries to prove the protocol's security by specifying insecure states and attempts to prove them unreachable. In order to prove unreachability of insecure states, the NRL protocol analyser performs exhaustive search backwards from the state or it uses proof techniques for reasoning about the state machine models. It uses automatic invariant generation to limit a potentially infinite search space in combination with exploration of the remaining space to generate attacks on insecure protocols. This provides security proofs for secure ones, even in the face of a potentially unlimited number of protocol executions or an unlimited number of intruder actions. To handle unbounded state space, the NRL protocol analyzer uses narrowing for the words those obey reduction rules. It also uses induction to prove that infinite sets of states are unreachable. Therefore NRL protocol analyzer can be used to prove security properties of cryptographic protocols and also locate security flaws. NRL protocol analyzer has been successfully applied to find previously unknown flaws in the Simmons selective broadcast protocol [26] and Burns-Mitchell resource sharing protocol [2] and some hidden assumptions in the Neuman-Stubblebine reauthentication protocol [25] and the Aziz-Diffie wireless communication protocol [1].

2.8.2 Failure Divergence's Refinement (FDR) Checker

FDR [73] is a model-checking tool that was built based on the theory of concurrency and it uses the Communicating Sequential Processes (CSP) to model the security protocol. FDR primarily checks the state machine's determinism in order to verify security properties. Its method of establishing whether a property holds is to test for the refinement of the transition system capturing the property by the candidate machine. Therefore technically FDR is a refinement checker even though it is described as a model checker. FDR converts two CSP process expressions into Labelled Transition Systems (LTSs), and then determines whether one of the processes is a refinement of the other within some specified semantic model (traces, failures, failures/divergence and some other alternatives). It has been proven that modelling in CSP is tedious and error prone. Expertise as well as lot of time is required to produce a CSP description of a protocol. In order to make things simpler, Lowe built Casper (A Compiler for the Analysis of Security Protocols) [55] which is a modelling tool and that can generate a CSP code of a protocol from a lower level of abstraction it. FDR can then be used to check this auto-generated CSP code.

The advantage of Casper is that it allows the protocol designer to specify the properties of the protocol using the Alice-bob notation which in a way helps in drastically reducing the reduces the complexity of verification. When used with Casper, FDR is able to generate attacking runs that can comprehend any prospective security flaws. The main aim of using FDR in the case of protocol authentication is to discover whether there is any specific attack within a protocol run especially in scenarios wherein an intruder is trying to masquerade as another one. If an attack is found the protocol is subjected to analysis to find the potential flaw. After removing the potential flaw FDR is further used to verify that there are no more attacks and this is done by running the protocol on a small scale system. Though it has been proven that FDR can be rather fast on small systems its effectiveness on large scale systems is yet to be explored. FDR has some user-specified limitations as to the number of objects that can be considered and therefore failure to find an attack is also restricted within those limits. G. Lowe demonstrated successfully the man-in-the-middle attack in Needham-Schroeder protocol [69], after 17 years of its publication, which really inspired many people to start working in the direction of authentication of security protocols..

2.8.3 Longley-Rigby Tool

Longley and Rigby [52] developed a rule based system to find subtle flaws in a given protocol. It constructs a tree in which each node represents a data item and child nodes represent data items that are required for representation of the data at parent node. The root node represents data that is required by the intruder. Here the goal is to verify the possibility of reaching the root node by discovering the knowledge at each leafs. This tool has been applied to hierarchical key management schemes successfully to disclose subtle and previously unknown flaws. The distinguish feature of Longley Rigby tool is it allows human intervention. Once the tool comes to conclusion that a word cannot be determined by the intruder, then the user can intervene to verify the possibility. If the word is judged to be accessible, then that word can be inserted into the database and the search operation is continued.

2.8.4 ProVerif

ProVerif [19] tool was developed by Bruno Blanchet. It uses *Pro-log* rules to represent facts and rules and an efficient algorithm to determine whether a fact can be proved from these rules or not. It also provides cryptographic primitives like digital signatures, hash functions, symmetric & asymmetric cryptography, signature proof of knowledge and bit-commitment. The tool is having the ability to evaluate properties like reachability, equivalence and correspondence assertions. These characteristics are specifically useful to analyze security protocols that are designed to ensure secrecy and authentication. Emerging properties such as privacy, traceability and verifiability can also be taken into consideration. The tool is capable of analyzing given protocols under unbounded number of sessions and message space. It can also reconstruct an attack when a property of a given security protocol cannot be proved. This feature will be very much helpful to the analyst/ developer of the protocol to carryout the modifications/ corrections such that an attack can be nullified.

ProVerif takes a model of cryptographic protocol and security properties which needs to be verified as input. The security properties are modelled as derivability queries. It translates the input into a set of Horn classes, which are later analysed by resolution algorithm. The tool relies on the fact that the translation into Horn clauses is an approximation. This approximation is sound, meaning that, if an attack exists in the cryptographic protocol, then it also exists in the approximation. It is incomplete, because if an attack is found in the approximation, it does not guarantee that it also exists in the cryptographic protocol. It uses unification, to avoid the problem of state space explosion. The advantage of this tool is that it requires minimum number of resources to analyse many protocols including that of *Skeme*.

2.8.5 Scyther Tool

Scyther tool [33] was designed based on pattern refinement algorithm to provide precise representations for sets of traces. Scyther tool characterize the protocols, yielding a nite representation of all possible protocol behaviors. This feature helps it in analysis of various types of attacks, different behaviors and also to verify correctness of a protocol for an unbounded number of sessions. It provides an option to perform multi-protocol analysis. Scyther tool is incorporated with number of new features to provide best performance results. At any cost Scyther guarantees termination of protocol under unbounded number of sessions and also produces the output in the form of proof tree by making use of backend.
Unlike other unbounded verification tools, Scyther tool provide some useful information even in the absence of attack and no possibility of unbounded correctness. When the above one is the case then results must be interpreted in similar to bounded verification tools. Scyther also provides GUI (Graphical User Interface)

to analyze different classes of protocol behaviors or attacks. Using Scyther tool one can perform multi protocol analysis in which parallel composition of tow (sub) protocols can be analyzed [31]. This type of facility was not there with other tools because of state space exploration problem. Multi protocol analysis has become feasible with Scyther tool because of its state-of-art performance improvement in analysis.

2.8.6 AVISPA Tool

AVIPA [9, 80] provides a formal language to specify the security properties of authentication protocols. The language provided by AVISPA is expressive and modular in nature and is known by the name HLPSL (High-Level Protocol Specification Language). With the help of HLPSL, complex security properties can be expressed which includes defining various cryptographic primitives as well as their algebraic properties. It also helps in modelling different intruder models as well as in the specification of different control flow patterns. At the back-end AVISPA makes use of a variety of state-of-the-art model checkers that helps in the automatic verification of security protocols. All these properties makes AVSPA well suited for specifying modern industrial-scale protocols. Having outlined a number of tools that could be used for validating security protocols, as mentioned above, we have come to the conclusion of using the AVISPA tool to validate DNSSEC security parameters i.e *data origin authentications* and *data integrity.* Table 2.2 shows the various prominent automatic tools that are used to validate security protocols. The tools are classified based on the representation of the protocol and the method followed to analyse it.

	Model Checking		Theorem Proving
Symbolic	NRL FDR AVSPA	Scyther ProVerif AVISPA (TA4SPA)	Isabelle/ HOL
Cryptographic	Unbounded $\longrightarrow$	Crypto Verif	BPW (in Isbelle HOL) Game based security Proof (in Coq)

Table 2.2: Categorization of Tools for cryptographic protocol analysis

2.9 Performance Enhancement Methods for DNSSEC Protocol

DNSSEC protocol protects DNS system, by verifying *data origin authentication* and *data integrity* of DNS responses. If such functionalities are to be provided certainly there will be overhead in the present DNS system. The overhead from client's perspective is that client needs additional information to verify authentication and integrity. In order to get additional information client has to send more number of queries and receive responses from servers and also it has to perform additional operations like storing, comparing etc. From server's perspective, each server has to maintain additional information regarding all of its immediate descendant servers, which are responsible for sub-domains (zones). Each server should also maintain extra information regarding the systems under its own jurisdiction. From network point of view it has to handle additional traffic (extra requests/ responses that are sent) that gets added to the network. In the following subsections we discuss some of the methods proposed in the literature to improve the performance of DNSSEC protocol.

2.9.1 DHT-based Architectures

Deployment of DNSSEC for large Internet Service Provider (ISP) will have profound impact on performance. It seems that at least 5 times more nodes are required if one wishes to migrate the DNS resolving platforms for the existing architecture. There is a proposal for an alternative architecture wherein the DNS traffic can be split between the nodes. To achieve this rather than using the IP addresses of the queries the splitting of nodes is done according to FQDN, i.e., Fully Qualified Domain Names. This will result in the usage of 30% less nodes in such architecture. One drawback of such an architecture is that the nodes of the platform will not be having a uniform distribution of the resources. Moreover, there is always a reluctance from the operational teams to do any modification with the current load balancing infrastructure. Investigations are going on to find out how to do resource optimization on an ISP operational DNSSEC resolving platform. To this end, some work has been carried out over a Distributed Hash Table (DHT) protocol [63] to demonstrate the effectiveness of pro-active caching. The reduction in the number of nodes achieved is claimed to be 3.5 times with this kind of approach.

2.9.2 PREFETCHing

The design of DNS resolving platforms are meant for quick and light resolutions over the internet. It has been demonstrated through experiments [62] that at least 4 times more CPU time is required with DNSSEC resolutions as compared to that of DNS. This is due to the larger payload and signature checks that the DNSSEC has to carry out as compared to that of DNS. $PEEFETCH_X$ [64] is an alternative architecture that helps to solve the issue related to resolving platforms increasing their size during DNSSEC migration. The PREFETCH$_X$ architecture, designed to: 1) Reduce the number of nodes (i.e. CPU or resources). 2) Provide management facilities for Operation, Administration and Management (OAM) such as overcoming nodes fail-over or addition of a node. 3) Deliver limited impact on the core network with a straight forward design to ease its deployment. Currently, according to the IP addresses, the traffic is split between the nodes by the existing resolving platforms like IP_{XOR}. PREFETCH$_X$ requires less nodes as compared to IP_{XOR} because it makes use of the Zipf function which in turn uses a layered sharing mechanism between the nodes. Zipf is FQDN's popularity distribution and requires 4 times less nodes as it uses a cache and cache sharing mechanism. The network infrastructure is not affected by PREFETCH$_X$ and therefore it can be deployed with ease. As it is possible to define 'X', i.e., the number of prefetched FQDNs, it is possible to make PREFETCH$_X$ suitable for different types of traffic and thus makes it highly scalable.

2.9.3 Optimizing Negative Caching

Negative Caching for DNSSEC-Oblivious resolvers (NCDO) [86] was proposed to improve DNSSEC protocol performance indirectly by preventing Denial of Service (DoS)/ Distributed Denial of Service (DDoS) attacks. Attackers send queries to the servers with names who's information is not available so frequently to create a kind of DOS attack. Full Resolvers can prevent such attacks by caching negative results and then preventing of forwarding the negative queries to the authoritative servers. NCDO method utilizes (Next Secure) NSEC/ NSEC3 resource records to gather non-existence name information and store it in cache memory, so that the negative queries can be answered locally. By improving negative cache hit rate, response time can be significantly improved in most cases. NCDO is light-weighted as compared to DNSSEC because it strips away some operations such as key management, zone signing and record authentication. Trace-driven simulations show the effectiveness of NCDO in promoting negative cache hit rate. Table 2.3 shows the characteristic and performance comparison among DNS, NCDO and DNSSEC.

	NSEC/ NSEC3	Signing & authentication	Deployment & operation cost	Cache hit rate	Response time	Cache consistency	Security level
DNS	No	No	Low	Low	Moderate	Low	Low
NCDO	Yes	No	Moderate	High	Fast	High	Low
DNSSEC	Yes	Yes	High	Low	Slow	Low	High

Table 2.3: Characteristic and Performance Comparison among DNS, NCDO and DNSSEC

2.9.4 Negotiating DNSSEC Algorithms

In order to ensure security and efficiency from cryptographic protocols, there should be some provision within the protocols, that permits principals to negotiate the use of the *best cryptographic algorithms.* This process is usually referred to as cipher-suite negotiation, and it plays a vital role in most prominent protocols such as Transport Layer Security (TLS), Internet Protocol security (IPSec) and so on. However, such negotiation process is missing, particularly in some protocols which are designed for the distribution of cryptographically-signed objects, such as DNSSEC. There could be many reasons for not giving the importance to the negotiation process, but there will be an impact on performance of such protocols. Here a new Negotiating DNSSEC Algorithm [47] was proposed to improve the security and performance of DNSSEC protocol. This method is also compatible with intermediate proxies and legacy proxies. It is shown that significant improvement would result in performance and security, if more domains support multiple algorithms. And it is also mentioned that the same idea can also be applied to Wireless Sensor Networks (WSN) systems to achieve secure and efficient distribution of signed data.

2.9.5 An Advanced Client Based DNSSEC Validation

DNSSEC is designed to protect DNS system, however DNSSEC has not yet been deployed in complete name space of DNS system. This is because DNSSEC protocol imposed heavy workload and high administrative cost on full resolvers and servers respectively. Additionally, DNSSEC also did not concern about the security between DNS full resolver and client. Therefore there is a strong requirement of a complete solution to solve the issues mentioned above. The proposed solution that is *An Advanced Client based DNSSEC Validation* [50] provides a new validation mechanism with acceptable workload on DNS full resolver and client. It also extended the DNSSEC security mechanism between client and full resolver. According to the results of preliminary evaluations it is confirmed that it is possible to reduce the workload of DNS full resolver by transferring the DNSSEC validation process to clients with acceptable extra workload.

2.10 Summary

In this Chapter we gave an overview of the foundational concepts and related work on which this thesis is built. We started with a discussion on the Domain Name System and explained in detail the working principles of DNS. We gave a brief overview of DNSSEC and moved on to describe about the importance of formal methods in the specification and verification of authentication protocols. We discussed on Modal logics like BAN, GNY, AT, VO, SVO etc. that are used for reasoning about security protocols. The necessity of automation tools with respect to validation of security protocols is discussed. Various performance enhancement techniques like Advanced Client Based DNSSEC Validation, Negotiating DNSSEC Algorithms, DHT-based Architectures etc, with respect to DNSSEC protocol are discussed.

Chapter 3

Formal Analysis of DNSSEC Protocol

It is well known that security protocols are widely used to provide security services in different distributed systems. Flaws in the design of these protocols could have negative consequences over the systems they are supposed to protect. There is a strong requirement for specification and verification of these protocols before using them. Logic-based Specification and Verification is one of the widely used formal techniques in the domain of security protocols and many modal logic based systems like BAN, GNY and SVO have been out into use. The main objective of applying formal methods to verify security protocols is to detect the *logical flaws* if any in their design. In this Chapter the research problem that is addressed is related to *formal reasoning of DNSSEC (Domain Name System Security Extensions) protocol*, which was intended to provide security to DNS (Domain Name System) protocol. [1]

As human beings, we remember names much better than numbers. But to access any system that is connected to the *Internet*, we should know the Internet protocol (IP) address of the corresponding system. To overcome the difficulty of remembering all IP addresses, the Internet Engineering Task Force (IETF)

[1]A part of this chapter is published as *Reasoning about DNSSEC*, Kollapalli Ramesh Babu, Vineet Padmanabhan, and Wilson Naik Bhukya, in Sombattheera et al. [2], *MIWAI 2011*: pp. 75 - 86. [**DBLP indexed**]

provided an Internet service called DNS [67, 66, 68]. As mentioned previously in Chapter 2 the DNS service provides the mapping of a given domain name to its corresponding IP address and vice versa. Name servers and resolvers form two major components of a DNS system wherein name servers take care of returning the information about domains whenever it is queried by DNS resolvers. To service DNS requests from applications queries could be send to more than one name servers by DNS resolvers. Resource records are interpreted as correct as long as a name server is considered to be authoritative for a particular domain. Caching of resource records is done by resolvers so as to reduce server load as well as to improve availability. Cached records are not considered to be authoritative as they might contain incorrect information and there is no way to distinguish this from a user point of view. The common practise is that queries from the applications are send to stub resolvers which are in the form of library functions and these functions forward requests to standalone recursive resolvers. In order to answer any request these standalone resolvers perform multiple name server queries using cached responses. Internet Service Providers also use DNS servers with recursive, caching resolvers for their customers. The steps involved in a typical DNS look-up can be given as follows [30]:

1. A local stub server is invoked by a web browser which in turn will send a request to a local DNS server (a recursive, caching DNS resolver).

2. An IPv4 address for www.wikepedia.org is checked by the DNS server in its cache. A missing entry in the cache will prompt the DNS server to request a randomly chosen rot server for the lookup. The DNS servers have the IP addresses of the standard root servers builtin.

3. The name (IPv4 address) of the authoritative name server for the .org domain will be communicated by the root server in the form of a NS record. The root server will also append the IPv4 address record for the .org top level domain server.

4. A lookup for www.wikipedia.org is requested by the recursive DNS resolver

from the .org authoritative name server (as specified in the returned record by the root server).

5. The authoritative name server record for wikipedia.org will be send by the .org name server and will also append the IPv4 address record for the wikepedia.org authoritative name server.

6. The authoritative name server will be asked to look up for www.wikipedia.org by the recursive DNS resolver.

7. An authoritative answer for www.wikepedia.org is send by the name server of wikepedia.org

8. After adding the authoritative answer send by the name server to its cache, the DNS server returns a non-authoritative answer, i.e., the IPv4 address to the stub resolver.

3.1 Domain Name System Security Extensions (DNSSEC) Protocol

DNS system does not prevent attackers from modifying or injecting DNS messages. A typical attack, referred to as *DNS spoofing* [7], 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. Similarly any IP address can be taken over by an attacker who mounts a man-in-the-middle attack. It is possible for attackers to make machines connect to malicious hosts but at the same time can also completely secure the DNS data. 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 (DNSSEC) [5, 6, 10] was designed. Securing DNS means providing *data origin authentication* (i.e., ensuring the data is received from a party from which it was sent) and *data integrity* (i.e., ensuring the data is same as that 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 so that anyone in the Internet can see the DNS replies. In order to ensure origin authentication and integrity of data, all answers from DNSSEC enabled DNS servers are digitally signed. By checking the signature, a DNS client is able to check whether the information was originated from a legitimate server and whether the data is identical to the data on DNS server. To achieve security in Domain Name System, DNSSEC adds four records namely DNS Public Key (DNSKEY), Resource Record Signature (RRSIG), Delegation Signer (DS), and Next Secure (NSEC) [49] to the existing DNS System. These are explained as follows;

1. **DNSKEY:** DNSSEC uses public key cryptography to sign and authenticate DNS Resource Record sets (RRsets). The public keys are stored in DNSKEY resource records [49] and are used in the DNSSEC authentication process.
   An example DNSKEY record:
   test.com. 86400 IN DNSKEY 256 3 5 (public key value in base64 form)

2. **RRSIG:** The digital signatures of DNS replies are stored in RRSIG record. RRSIG value [49] is created by encrypting the hash value of the DNS reply with private key of the zone.
   An example RRSIG record:
   test.com. 86400 IN RRSIG A 5 3 86400 20090507235959 (20090501000000 41148 test.com. encrypted hash of the RRset)

3. **DS:** Contains the hash value [49] of a child zone's DNSKEY which is needed in authenticating client zone DNSKEY.
   An example DS record:
   test.com. 86400 IN DS 46894 5 1 (encrypted public key value in base64 form)

4. **NSEC:** The NSEC RR [49] points to the next valid name in the zone file and is used to provide proof of non-existence of any name within a zone. The last NSEC in a zone will point back to the zone root or apex.

An example NSEC record:

test.com. 86400 IN NSEC ns1.test.com. (A NS SOA MX NSEC)

The DNS server in a zone creates public key and private key pair and publishes the public key in the zone [5]. The private key is kept in secrecy. 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 zone's data but 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 zone's 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* [5]. 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 process is continued until the chain finally ends with a DNSKEY RR whose corresponding private key signs the desired DNS data.

For instance, consider the URL `"www.dcis.uohyd.ernet.in."`. Here the root DNSKEY RRset can be used to authenticate the DS RRset for the `"in"` domain. The `"in"` domain DS RRset contains a hash that matches some `"ernet"` domain DNSKEY and the `"ernet"` domain DS RRset contains a hash that matches some `"uohyd"` domain DNSKEY. This DNSKEY's corresponding private key signs the `"dcis"` DNSKEY RRset. Private key counterparts of the `"dcis"` DNSKEY RRset signs data records such as `"www.dcis.uohyd.ernet.in."`.

3.1.1 DNSSEC Work Flow

DNSSEC protocol works in client - server scenario. When the client sends a request to the DNS Server, additional data is added by the DNS server to the protocol responses of the DNS and thereby provides extra information. This extra information (RRSIG, DNSKEY) allows the client to authenticate the RRset data response[2]. The hash value of the data can be generated by the client by taking the RRset response and making use of the algorithm referenced in the RRSIG record. The DNSKEY public key can be used to encrypt the RRSIG value which in turn will help in decrypting the hash value in the RRSIG record. This operation will enable the client to check whether the hash value of the RRset data and the decrypted RRSIG hash value matches or not. Usually a DNSSEC response will be having an additional section that provides the DNSKEY. The client needs to validate the DNSKEY within a limited time period failing which he/she also needs to validate the DNSKEY value. By doing this it is implied that the RRSIG record on the DNSKEY value is verified and the procedure to be followed for doing this is the same as that which is used for RRset validation.

It should be kept in mind that the validation of a domain zone key implies that there is a chain of trust construction back to a trust anchor point. The domain key needs to be a trust anchor and if it is not the case then the client needs to query the parent zone. This is needed to get the DS record of the child zone and by querying the parent zone, DNSKEY RR, RRSIG(DS) RR, and DS RR is returned. The DNSKEY of the parent zone is used to validate the DS RR. The parent zone public key must also be validated. This process of validating the resource records in an iterative manner leads to the chain of trust construction which should eventually lead to a trust anchor. Validation of DNS response can be considered to be successful at that point of time. Figure 3.1 explains how the authentication chain will be established using DS RRset and DNSKEY RRset. Consider that the host `dcis` in `uohyd` domain has made a DNS Request for `D.com` whose RRset is available in `com` domain, then the name resolution process involves the following steps.

[2]RRset = It is a collection of RRs in a DNS ZONE that share a common name, class and type.

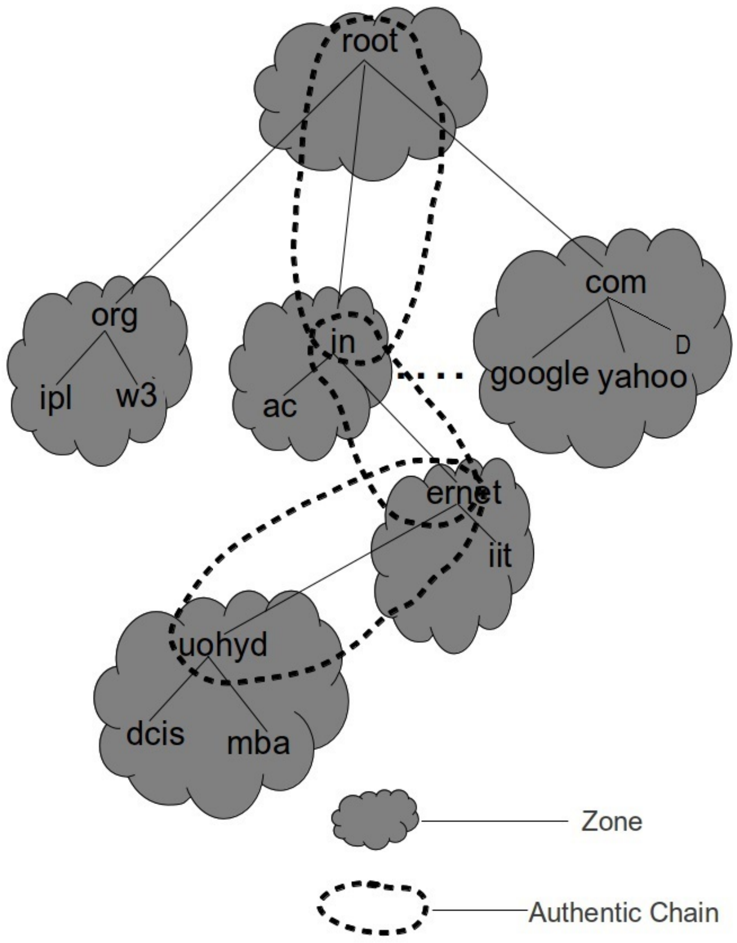


Figure 3.1: A sample scenario of DNS

M_{cs}: *dcis* $\longrightarrow$ *uohyd* :$\langle DNSReq., D.com.\rangle$

M_{cs}: *uohyd* $\longrightarrow$ *ernet* :$\langle DNSReq., D.com.\rangle$

M_{cs}: *ernet* $\longrightarrow$ *in* :$\langle DNSReq., D.com.\rangle$

M_{cs}: *in* $\longrightarrow$ *root* :$\langle DNSReq., D.com.\rangle$

M_{cs}: *root* $\longrightarrow$ *com* :$\langle DNSReq., D.com.\rangle$

M_{sc}: *com* $\longrightarrow$ *root* :$\langle RRset_{D.com.}, RRSIG_{D.com.}, DNSKEY_{com}\rangle$

M_{sc}: *root* $\longrightarrow$ *in* :$\langle RRset_{D.com.}, RRSIG_{D.com.}, DNSKEY_{com}\rangle$

M_{sc}: *in* $\longrightarrow$ *ernet* :$\langle RRset_{D.com.}, RRSIG_{D.com.}, DNSKEY_{com}\rangle$

M_{sc}: *ernet* $\longrightarrow$ *uohyd* :$\langle RRset_{D.com.}, RRSIG_{D.com.}, DNSKEY_{com}\rangle$

M_{sc}: *uohyd* $\longrightarrow$ *dcis* :$\langle RRset_{D.com.}, RRSIG(RRset_{D.com.}), DNSKEY_{com}\rangle$

where M_{cs} stands for Message sent from client to server, M_{sc} stands for Message sent from server to client and $\langle DNSReq., D.com.\rangle$ stands for DNS Request for `D.com` domain. After receiving the DNS Response form *uohyd* server, *dcis* host will try to construct `chain of trust` between the source of response and trust of anchor as follows.

M_{cs}: *dcis* $\longrightarrow$ *uohyd* :$\langle DSReq., com.\rangle$

M_{cs}: *uohyd* $\longrightarrow$ *ernet* :$\langle DSReq., com.\rangle$

M_{cs}: *ernet* $\longrightarrow$ *in* :$\langle DSReq., com.\rangle$

M_{cs}: *in* $\longrightarrow$ *root* :$\langle DSReq., com.\rangle$

M_{sc}: *root* $\longrightarrow$ *in* :$\langle DS_{com}, RRSIG(DS_{com}), DNSKEY_{root}\rangle$

M_{sc}: *in* $\longrightarrow$ *ernet* :$\langle DS_{com}, RRSIG(DS_{com}), DNSKEY_{root}\rangle$

M_{sc}: *ernet* $\longrightarrow$ *uohyd* :$\langle DS_{com}, RRSIG(DS_{com}), DNSKEY_{root}\rangle$

M_{sc}: *uohyd* $\longrightarrow$ *dcis* :$\langle DS_{com}, RRSIG(DS_{com}), DNSKEY_{root}\rangle$

where $\langle DSReq., com.\rangle$ stands for Delegation Signer Request for `com` domain. Now, origin authentication and integrity of data is checked as follows.

3.1.2 Origin Authentication and Data Integrity

Authentication will be achieved by establishing authentication chain between the zones with the help of DS RRset and DNSKEY RRsets. For instance, in our example `"dcis"` already knows the authentication public key(DNSKEY) of root. To make newly learned DNSKEY authentic, it needs to establish an authentication chain back to known DNSKEY i.e. root's DNSKEY. The steps given below show how this chain of trust helps in verifying the Origin of the response.

1. Verifying Sender DNSKEY (i.e. DNSKEY$_{com}$)

    (a) $DNSKEY_{root}\{RRSIG(DS_{com})\} == hash(DS_{com})$

    (b) $DS_{com} == hash(DNSKEY_{com})$

    Initially, the sender's (i.e., .com's) DNSKEY has to be verified (1). In order to do this, the hash value of the DS record of .com should match the decrypted value of RRSIG of DS_{com} as shown in (a). Finally, the hash value of the DNSKEY of .com is compared with DS_{com} record as given in (b). In the next step, we have to verify the DNSKEY of parent zone of .com which is `Root` as given below.

2. Verifying Parent Zone DNSKEY(i.e. DNSKEY$_{root}$): The DNSKEY of root zone is believed by every one. Therefore, it is not required to be verified and from this we can get authentication chain to the root node.

Data Integrity will be achieved if the following condition is true

$$hash(RRset_{D.com}) == \{RRSIG_{D.com}\}DNSKEY_{com}$$

i.e., first, the hash value is computed for the received RRset. Secondly, RRSIG value is decrypted using DNSKEY and finally both the values are compared. If both values are equal, then it implies that the received data is correct. If not, we can arrive at the conclusion that it was modified inbetween.

3.2 Formal Representation of DNSSEC Protocol

The representation of a protocol using Alice Bob (A - B) notation as described in Chapter 2 will help us in the better understanding of the protocol operations and analysis as well. Table 3.1 shows the constructs used in representation of the DNSSEC protocol.

3.2.1 Constructs used in DNSSEC protocol representation

Term	Description
$DNS\_request \longleftarrow$ $Fully\ Qualified\ Domain\ Name\ (FQDN) \mid IP\_address$	DNS_request holds either FQDN or IP address.
$RRSet\_S \longleftarrow$ $IP\_address \mid FQDN$	RRSet_S holds either IP address or FQDN.
$RRSIG[RRSet\_S] \longleftarrow$ $hash(RRSet\_S)_{PK[S,K]^{-1}}$	RRSIG[RRSet_S] holds hash value of RRSet_S encrypted with private key of server (S).
$DS\_S \longleftarrow hash(PK[S,K])$	DS_S holds hash value of the public key of server (S).
$RRSIG[DS\_S] \longleftarrow$ $hash(DS\_S)_{PK[R,K]^{-1}}$	RRSIG[DS_S] holds hash value of DS_S encrypted with private key of root (R).
$DNSKEY\_S \longleftarrow PK[S,K]$	DNSKEY_S holds the public key (K) of server (S).
$DNSKEY\_R \longleftarrow PK[R,K]$	DNSKEY_R holds the public key (K) of root (R).
$PK[S,K]$	K is a public key of server (S).
$PK[R,K]$	K is a public key of root (R).

Table 3.1: Constructs used in DNSSEC protocol representation

3.2.2 DNSSEC Protocol in (A - B) Notation

In this notation C stands for client, S for server and R for Receiver.

```
1.  C  ->  S:  DNS_request

2.  S  ->  C:  RRSet_S, RRSIG[RRSet_S], DNSKEY_S

3. C "verify" if  H(RRSet_S) = {RRSIG[RRSet_S]}DNSKEY_S then

       continue with step 4.

       else discard RRSet_S and exit.

4.  C  ->  R:  DS_request
```

```
5.  R  ->  C:  DS_S, RRSIG[DS_S], DNSKEY_R
6. C "verify" if  H(DS_S) = {RRSIG[DS_S]}DNSKEY_R then
       continue with step 7.
       else discard DS_S and exit
7. C "verify" if H(DNSKEY_S) = (DS_S) then
   accept RRSet_S.
   else discard RRSet_S and exit.
```

3.3 SVO Logic for DNSSEC Protocol

Modal Logic is an umbrella term for a class of logics extending classical propositional logic with modal operators that qualify the expressions in their scope. Accordingly, if we consider the standard modal operators of necessity ($\Box$) and possibility ($\Diamond$), and a propositional letter p, we can use p to express that the proposition represented by p is true, $\neg p$ to specify that the proposition represented by p is false (or that the opposite/negation of p is true), $\Box p$ to indicate the the proposition represented by p is necessary, and $\Diamond p$, meaning that the proposition represented by p is possible. A plethora of interpretations for the modal operators have been proposed to cover different concepts of necessity/possibility. For example, under an epistemic interpretation, $\Box$ and $\Diamond$ can be understood, respectively as knowledge and belief. Often different interpretations require to specify axioms and rules to model the behaviours and relationships among the modal operators. Several modal logics have been proposed in the field of security and access control to model various underlying security and access control protocols. Modal logic based tools allow us to describe the beliefs of parties in the protocol and to study the evolution of these beliefs as a consequence of communication. For example

- if you have sent Bob a number that you have never used for this purpose before, and if you subsequently receive from Bob something that depends on knowing that number, then you should believe that Bob's message originated recently (at least after yours)

- if you believe that only you and Bob know K, then you should believe that anything you receive protected with K comes originally from Bob
- if you believe that K is Bob's public key, then you should believe that any signed message that you can successfully verify with K comes originally from Bob
- ...

As discussed in Section 2.7 of Chapter 2 there are some advantages and at the same time some disadvantages in modal logics like BAN, AT and GNY. To overcome the disadvantages and maintaining advantages of BAN, AT and GNY logics, the SVO [79] logic was developed. SVO logic unifies the functionalities of BAN, AT and GNY logics. SVO logic adds some more functionalities to make it a preferable choice so as to apply it to analyze security protocols. The details of inference rules, and the axioms for reasoning about DNSSEC are discussed below. The inference rules are indexed by $F1 \dots Fn$ and axioms by $A1 \dots Am$, where n, m $\in$ I are integers. The language of SVO is developed for dealing with abstract protocols and therefore *message sending* is composed of *expressions* in the language and is able to cover public keys, functions and message comprehensibility. A set of primitive terms T consisting of sets of constant symbols are available for representing principals, shared keys, public keys, private keys etc. Though formulae and messages are segregated by having two formal languages, any reference to SVO is applicable to both whereas truth/falsity is attributed only to formulae. Formulae can also be attributed with a principal's belief and both messages and formulae are built from T. The language of messages has the following interpretation

- X is a message if $X \in T$
- $F(X_1, \dots, X_n)$ is a message if $X_1, \dots, X_n$ are messages and F is any function. $F\{X\}_k$ as well as $F[X]_k$ is also allowed to represent encryptions as well as signed messages.
- φ is a message if φ is a formula

The language of formulae is interpretd as follows:

- $P \xleftrightarrow{k} Q$, $Pk_{\psi}(P,k)$, $Pk_{\sigma}(P,k)$ and $Pk_{\delta}(P,k)$ are formulae when P and Q are principals and k is a key.
- $SV(X,K,Y)$ is a formula when X and Y are messages and k is a key.
- $P\ sees\ X$, $P\ received\ X$, $P\ says\ X$, $P\ said\ X$ and $fresh(X)$ are formulae when X is a message and P is a principal.
- $\neg\varphi$ (not-φ) and $\varphi \wedge \psi$ (φ and ψ) are formulae if φ and ψ are formulae. The otehr connectives can be defined in a similar manner.
- $P\ believes\ \varphi$ and $P\ controls\ \varphi$ are formulae when φ is a formula and P is a principal.

3.3.1 SVO Inference Rules

F1: Modus Ponens: From $\varphi\ and\ (\varphi \rightarrow \psi)\ infer\ \psi$

The modus ponens has two premises, one is a conditional claim $(\varphi \rightarrow \psi)$ and the other one is the antecedent (φ) of the condition. If both premises are true, then it can be inferred that the consequent (ψ) is also true.

F2: Necessitation: $\vdash \varphi\ infer\ \vdash\ P\ believes\ \varphi$

The $\Gamma \vdash \varphi$ means that φ is derivable from the set of formulae Γ and set of axioms. $\vdash \varphi$ means that φ is a theorem that can be derivable from axioms alone. The Necessitation rule tells if φ is a theorem then it can be inferred that $P\ believes\ \varphi$. The symbol $\vdash$ is called turnstile, assertion sign or assertion symbol. It is read as, yields, proves or entails.

3.3.2 SVO Axioms for Reasoning about DNSSEC

A1: **Distributive Axiom**

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

The axiom states that a principal P believes all that logically follows from it's beliefs are true.

A2: **Truth Axiom**

$P\ believes\ \varphi \rightarrow \varphi$

The axiom states that the beliefs held by principal P is true.

A3: **Positive Introspection**

$P\ believes\ \varphi \rightarrow P\ believes(P\ believes\ \varphi)$

The axiom states that a principal P can tell what it believes.

A4: **Negative Introspection**

$\neg(P\ believes\ \varphi) \rightarrow P\ believes(\neg P\ believes\ \varphi)$

The axiom states that the principal P cannot tell what it does not believe.

A5: **Received Axiom**

(a) $P\ received\ (X_1, \ldots, X_n) \rightarrow P\ received\ X_i, for\ i = 1, \ldots, n.$

The Principal P can concatenate the messages $X_1, \ldots, X_n$, if individual messages $X_1, \ldots, X_n$ are received.

(b) $(P\ received\ \{X\}_{K^+}\ \wedge\ P\ sees\ K^-) \rightarrow P\ received\ X$

In case of symmetric cryptography, $K^+ = K^- = K$. In case of asymmetric cryptography, K^+ is a public key and K^- is the associated private key.

A6: **Source Association**

$(PK_\sigma(Q, K) \wedge\ received\ X \wedge SV(X, K, Y)) \rightarrow Q\ \ said\ Y$

The keys and signatures are used to identify sender of the message. $(PK_\sigma(Q, K))$ represents the fact that K is the public signature verification key for the principal Q. Similarly, *SV(X,K,Y)* represents that, given a signed message X, by applying key K to it will verify the message Y which was signed with private key $\tilde{K}$.

A7: **Seeing Axiom**

(a) $P\ received\ X \rightarrow P\ sees\ X$

(b) $(P\ seesX1\ \wedge ... \wedge P\ sees\ X_n) \rightarrow (P\ sees\ F(X1, ..., X_n))$

The Principal P can see all the things that it receives. It can also see the messages that can be computed from the received messages.

A8: **Freshness Axiom**

(a) $(fresh(X_i) \rightarrow fresh(X1, ..., X_n)$

(b) $(fresh(X1, ..., X_n) \rightarrow fresh(F(X1, ..., X_n))$

fresh(X), means, that X has not been sent in any message prior to the current protocol run. The principal believes that the concatenated message is fresh if one of its message is fresh. The function F of the message is also fresh.

A9: **Saying Axiom**

(a) $P\ said(X_1, \ldots, X_n) \rightarrow P\ saidX_i\ \wedge P\ has\ X_i, for\ i = 1, \ldots, n.$

(b) $P\ says\ (X_1, \ldots, X_n) \rightarrow (P\ said\ (X_1, \ldots, X_n) \wedge P\ says\ X_i), for\ i = 1, \ldots, n.$

P said X, means, P has sent a message containing X at some prior point, and P *believed* X and understood that it was sending X at that time.

P says X, means, X is a message P said recently. P must have said X since the beginning of current epoch.

P has X, means, X is a message P can see. This may indicate: X initially available to *P*, X received by *P*, X freshly generated by P , and X can be constructed by P from the above.

If a concatenated message is said by a Principle, then it can be comprehended that principal also said individuals of that message. A Principal who recently says a concatenated message can be treated as a concatenated message said by the same principal and also individual message.

A10: **Jurisdiction and Nonce verification Axiom**

(a) $(P\ controls\ \varphi \wedge P\ says\ \varphi) \rightarrow \varphi$

(b) $(fresh(X) \wedge P\ said\ X) \rightarrow P\ says\ X$

Part (a) of axiom 10 says that φ is under the absolute control of the Principal and part (b) distinguishes a message from having been said (sometime) to having been said during the current interval of time.

A11: **Possession Axiom**

(a) $P\ received\ X \rightarrow P\ has\ X$

(b) $P\ has(X1, ..., X_n) \rightarrow P\ hasX_i$

(c) $(P\ has\ X_1 \wedge ... \wedge P\ has\ X_n) \rightarrow P\ has\ F(X_1, ..., X_n)$

The principal P *received* X, means, P is having X.

The principal P is having a set of messages $X_1, ..., X_n$ means P is having individual messages.

The principal P is having individual messages $X_1, ..., X_n$ and function 'F' then P is also having result of $F(X_1, ..., X_n)$.

3.3.3 Extended SVO logic for Reasoning about DNSSEC

We have extended SVO logic by adding two new axioms for reasoning about DNSSEC protocol.

A12: **Authentication Axiom:**

$(P\ believes\ DNSKEY\_root\ \wedge\ P\ believes\ Equal(DS\_S,$
$\{RRSIG[DS\_S]\}_{DNSKEY\_root})\ \wedge\ P\ believes\ Equal(DS\_S,$
$hash(DNSKEY\_S))\ \rightarrow\ P\ believes\ DNSKEY\_S$

This axiom is used to authenticate the public key of server(S). The DNSKEY record holds the public key, DS record holds hash value of the public key, RRSIG record holds the hash value of a record signed with private key of server. Equal function checks the equality of two given parameters.

The axiom states that principal P should believe public key of root and principal P should believe that DS_S record which was sent by root did not get altered and principal P should believe that hash value of DNSKEY_S is equal to DS_S value then principal P believes the DNSKEY_S.

A13: **Integrity Axiom:**

$(P\ believes\ DNSKEY\_S\ \wedge\ P\ believes\ Equal(hash(RRSet\_S),$
$\{RRSIG[RRSet\_S]\}_{DNSKEY\_S})) \rightarrow\ P\ believes\ RRSet\_S$

The axiom states that the principal P should believe the public key of server and principal P should believe that if the data (RRSet_S) sent by the server did not get altered then principal P should believe data (RRSet_S) sent by server.

3.4 Formal Analysis of DNSSEC

The importance of Modal logic and their wide usage in evaluating the security protocols were discussed in section 2.7 and it was shown that Modal logic systems like BAN, AT, GNY and SVO are widely used in reasoning about security protocols. These modal logic systems analyze protocols by deriving beliefs that honest principals hold. Here we are using SVO logic as outlined in 3.3 for reasoning about DNSSEC (Domain Name System Security Extensions) protocol.

3.4.1 SVO Logic Procedure to Analyze Protocol

In order to analyze a protocol in SVO logic the following steps are followed:

1. Write assumptions about initial state.

2. Annotate the protocol.

3. Assert comprehensions of received messages.

4. Assert interpretations of comprehended messages.

5. Use the logic to derive beliefs held by protocol principals.

In the first step we write initial conditions of the protocol, which includes the beliefs held by the principals, the possession of keys or any other prior assumptions the principals might be holding. It should be kept in mind that in DNSSEC the client establishes authentication chain back to the previously known DNSKEY. As noted earlier, since DNS is having a hierarchical structure wherein the trace goes back to the root key the initial assumption is that everyone believes the root zones DNSKEY. i.e.

A0 —*client believes* $DNSKEY_{root}$ which can be rewritten as

A1 —*client believes PK(Root,* $DNSKEY_{root}$*)*

i.e., client believes the public key of the root which is DNSKEY$_{root}$. Premises P1 to P5 as given below also falls into the category of initial assumptions. In the second step we write the messages that are exchanged between principals which includes the received messages, that is those messages that are not lost in transit. Premises P6 and P7 as given below represents the annotation. In the third step we assert that the messages that are received by the principals are comprehended (eg. decryption of received message by using a key that is known already) as shown in premises P8 and P9. In the fourth step we write the inferred messages by principals with the received messages and the knowledge that principals initially have as outlined below;

$$\begin{gathered} client\ received\ (DNSKEY_{Sender}) \wedge\ client\ believes\ fresh(DS_{Sender}) \wedge \\ (\{Hash(DNSKEY_{Sender})\} \equiv \\ DS_{Sender}) \rightarrow client\ believes\ fresh\ (DNSKEY_{Sender}) \end{gathered}$$

$$\begin{gathered} client\ received\ X \wedge\ client\ received\ SIG(X) \wedge \\ client\ believesfresh(DNSKEY_{Sender}) \wedge\ (\{SIG(X)\}_{DNSKEY_{Sender}} \equiv \\ Hash(X)) \rightarrow client\ believes\ X \end{gathered}$$

In the fifth step we use axioms and premises to derive the goals of the protocol.

3.4.2 Protocol Analysis

The following are the Set of premises derived from the operation of a protocol.

P1: C believes fresh(PK(R,K))

P2: R believes fresh(PK(R,K))

P3: S believes fresh(PK(S,K))

The above premises are derived based on the initial assumptions of the protocol. Client(C) believes public key K of Root(R). Similarly Root(R) and Server(S) believes their own public keys.

P4: C believes R controls (fresh(PK(R,K))

P5: C believes S controls (fresh(PK(S,k))

The above premises says that C believes R and S have control over freshness of the their public keys.

P6: C received $\{S, C, RRSet, SV(SIG(RRSet), PK_\sigma(S, K), RRSet), PK(S, K)\}$

P7: C received $\{R, C, DS_S, SV(SIG(DS_S), PK_\sigma(R, K), DS_S), PK(R, K)\}$

P8: C believes C received $\{S, C, RRSet, SV(SIG(RRSet), PK_\sigma(S, K), RRSet), PK(S, K)\}$

P9: C believes C received $\{R, C, DS_S, SV(SIG(DS_S), PK_\sigma(R, K), DS_S), PK(R, K)\}$

The above premises says that Client(C) believes that it received Resource Record Set(RRSet), signature verification of RRSet and public key K required to verify the signature of the message received from server(S). It also believes that it received Delegation Signer (DS_S) record, corresponding signature verification record $SIG(DS_S)$ and the public key K required to verify the signature of the message received from Root(R).

3.4.2.1 Objective 1: Verifying Origin Authentication

To say DNSSEC achieves Origin Authentication we need to prove the rule given below

$$client\ believes\ DNSKEY_{sender}\ i.e.\ \texttt{"dcis"} believes\ DNSKEY_{com}$$

i.e if client believes in sender zones DNSKEY then it will believe the public key signature made by the sender using private key which can be verified using corresponding public key i.e. zone DNSKEY. To prove the above one we start with *initial assumptions* of the DNSSEC protocol. According to DNSSEC protocol to provide origin authentication it is assumed that all the nodes (systems) in domain name space believes the root node's public key by default. When this assumption is applied to our example then it can be formally written as given below

I1 —*dcis believe* $DNSKEY_{root}$

To verify "com" zone's DNSKEY, `"dcis"` sends a request to root i.e. com zone's parent for DS record set and `"dcis"` receives DS record set from root, which can be written as

$$dcis\ received\langle DS_{com},\ RRSIG(DS_{com}),\ DNSKEY_{root}\rangle$$

using A5.(a), we can derive the following assertions.

O1 —*dcis received* DS_{com}

O2 —*dcis received RRSIG(*DS_{com}*)*

O3 —*dcis received* $DNSKEY_{root}$

When `"dcis"` receives this DS Record Set, it compares it with

$$\{RRSIG(DS)\}_{DNSKEY_{root}}$$

If the data has been received without any modifications in between, then

O5 —$DS_{com} \equiv \{RRSIG(DS_{com})\}_{DNSKEY_{root}}$

And if the Above condition is satisfied then dcis believes DS$_{com}$ is fresh

O6 —$dcis\ believes\ fresh(DS_{com})$

O7 —*dcis received DNSKEY$_{com}$*

From O7,A7(b)

O8 —*dcis sees hash(DNSKEY$_{com}$)*

O9 —$DS_{com} \equiv hash(DNSKEY_{com})$

And if the above condition is satisfied, then *dcis* believes $DNSKEY_{com}$ From O9,O6,A1 we can get

O10 — ***dcis believes DNSKEY$_{com}$***

First, we have listed all the received data from both the *root* node and the *com* node. In order to authenticate the data received from *com* node, first we check whether the received data is altered during transit or not. Once it is confirmed that the received data was not altered, then we compute the hash value of the public key (DNSKEY) of *com* node. Then we compare this hash value with the hash value stored in Delegation Sign record (DS record), which has come from *root* node. If these two hash values match, then we believe the public key (DNSKEY) of *com* node. Since we believe the *root* whatever data sent by *root* node is also believed and therefore the objective of `data origin authentication` is proved.

3.4.2.2 Objective 2: Verifying Data Integrity

When `"dcis"` sends a request to uohyd for D.com. name resolution, and receives DNS reply which contains the naming resolution *RRset*, Digital Signature *RRSIG* and *DNSKEY* of com zone. Then it is proved that (In Origin Authentication Check)

dcis believes DNSKEY$_{com}$ which can be rewritten as

N1 —*dcis believes PK(com,DNSKEY$_{com}$)*

N2 —*dcis received* $\langle RRset_{D.com.}, RRSIG_{D.com.}, DNSKEY_{com} \rangle$

By applying A5.(a) to N2, we can get the following assertions.

N3 —*dcis received* $RRSET_{D.com}$

N4 —*dcis received* $DNSKEY_{com}$

N5 —*dcis received* $RRSIG_{D.com}$

If DNSKEY of received zone is fresh then

N6 —*Fresh(DNSKEY*$_{com}$*)*

In order to verify integrity, `dcis` compares calculated RRSET's hash value with

$$\{RRSIG_{D.com}\}_{DNSKEY_{com}}$$

If the data is received without any alterations, then

N7 —$hash(RRSET_{D.com}) \equiv \{RRSIG_{D.com}\}_{DNSKEY_{com}}$

From N2,N3, N6, N7 and A1, we can get

N8 — *dcis believes RRSET*$_{D.com}$

In section 3.4.2.1, it is already proved that *dcis believes DNSKEY*$_{com}$, now we use this fact to prove the second objective of data integrity. First we list the data that is received from the *com* node, then we compute the hash value of the received data (i.e $RRSET_{D.com}$) and compare with the hash value decrypted from $RRSIG_{D.com}$ record using public key ($DNSKEY_{com}$) of *com* node. Once these two hash values are matched then we can say that the data received from the *com* node was not modified in transit and hence the `data integrity` objective is proved.

3.5 Summary

Logic based frameworks are widely used in the specification and verification of security protocols. In this chapter we proposed a formal framework for analysing the DNSSEC protocol. We gave an in-depth analysis of DNSSEC to start with and used SVO logic for formally representing DNSSEC and also specifying the nuances involved in DNSSEC. Thereafter we proposed an extension of DNSSEC wherein two new axioms were added to the original SVO logic. Formal Analysis of DNSSEC in the new set up was carried out by verifying origin authentication and Data integrity with the help of message passing examples. By making use of inference rules as well as axioms of SVO logic we were able to successfully derive proof for *Data origin authentication* and *Data integrity* of DNSSEC protocol.

Chapter 4

Validating DNSSEC protocol

The usage of Internet and network based services have become very common these days. Some of the services are very critical and require robust security to avoid intolerable consequences. As we all know, to provide such services, a robust cryptographic security protocol must be used. The designing of such a secure protocol is always a challenging task. The complexity of security protocols is increasing day by day, due to their functionalities and type of services that they provide to protect the resources from unauthorised access. As the complexity grows, it becomes a very difficult and tedious task to validate its correctness manually. Therefore we need some automatic mechanism to validate the correctness of a given security protocol. The research problem that we address in this chapter is related to the validation of the security protocol DNSSEC using the automated verification tool called AVISPA"[1]

4.1 Security Protocol Verification Tool: AVISPA

Automated Validation of Internet Security Protocols and Applications (AVISPA) [9] [80] is an automatic formal validation tool used to validate correctness of security properties of Internet security protocols. AVISPA tool is aimed at, countering

[1]A part of this chapter has been accepted as *Automated validation of DNSSEC* in the International Conference on Computing Analytics and Networking (ICCAN 2017), 15-16 December 2017, Odhisa.

two types of attacks - one is the *Man-in-the-middle* attack and the other one is *replay* attack. AVISPA uses a High Level Protocol Specification Language (HLPSL) to write a protocol specification. HLPSL [28, 11] is a very expressive and intuitive language to model any security protocol. The architecture of AVISPA tool is as shown in Figure 4.1.

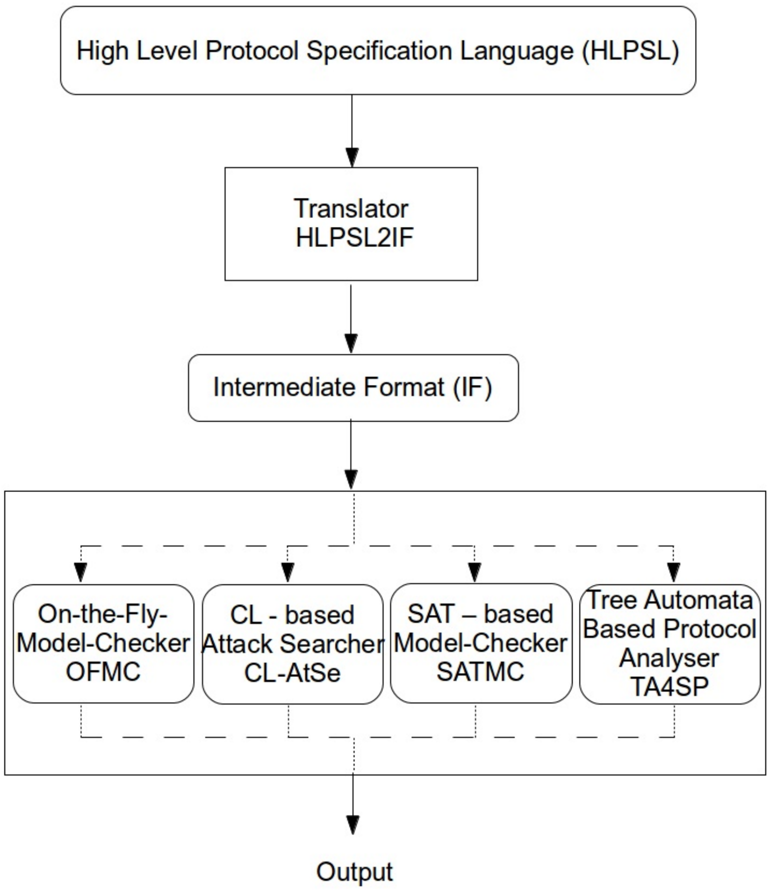

High Level Protocol Specification Language (HLPSL)
Translator HLPSL2IF
Intermediate Format (IF)
On-the-Fly-Model-Checker OFMC
CL - based Attack Searcher CL-AtSe
SAT – based Model-Checker SATMC
Tree Automata Based Protocol Analyser TA4SP
Output

Figure 4.1: Architecture of AVISPA Tool

The AVISPA tool accepts input specification in HLPSL and translates HLPSL specifications into an Intermediate Format (IF). IF is an intermediate step where the rules are re-written such that it will be more comfortable/ convenient for further processing it by back-end analyzers. The IF format of a protocol gets executed completely if no loop exist, otherwise it gets executed for a finite number of times. Finally, the execution may end-up with either by an identification of an attack on a given protocol or showing a proof that the protocol is safe within a finite number of sessions. AVISPA currently provides supports for four back-end analyzers. We give below a short description of each of the analysers.

4.1.1 On-the-Fly Model-Checker (OFMC)

The two major ideas that OFMC [16] uses to analyze security protocols are lazy and demand-driven searches. The infinite state space associated with a protocol is modeled using lazy data-types. Lazy data-type builds data without verifying the arguments. OFMC make use of this feature to specify and compute very large amount of data that generates arbitrarily. OFMC uses lazy evaluation technique to make search and heuristics independent from the model to which it is applied. It also applies the same technique to build infinite tree in a demand-driven fashion.

OFMC applies lazy intruder technique combined with lazy infinite space approach to reduce the search space by explicitly eliminating the intruder messages. OFMC makes use of infinite tree data-type supported by lazy programming language to implement infinite-state approach [15, 14]. This results into a finite and computable model using which OFMC can generate arbitrary prefixes of the tree on the fly. To search an infinite tree OFMC applies iterative deepening technique. Whenever an attack is found by OFMC then it returns in the form of attack-trace, which is a sequence of messages that are exchanged and leading to an attack-state. OFMC uses an optimization technique called lazy intruder to reduce the search tree significantly without eliminating any attacks. Whenever the receiver receives message in which some part is irrelevant, then the receiver will not further analyze the value of a particular message part. The lazy intruder exploits the above point to postpone the decision during search.

The lazy intruder is an optimization technique that significantly reduces the search tree without excluding any attacks. The lazy intruder technique exploits the fact that the actual value of some parts of a message is often not relevant for the receiver. Therefore, when the message or part of the message is not analysed by the receiver, then it gets replaced with a variable and the decision about it gets postponed during search. OFMC uses general search reduction technique called constraint differentiation technique for reducing search time. Many efficient search heuristic techniques are implemented by OFMC to reduce the search space.

4.1.2 SAT-based Model-Checker (SATMC)

SATMC [8] starts analyzing the problem by considering the bounded number of scenarios with a finite number of sessions. SATMC considers Dolev-Yao model for exchange of messages between principals, where the presence of intruder is also considered into account. SATMC first reduces the problem of 'checking whether a protocol is vulnerable to attacks within bounded number of scenarios' into satisfiability of a propositional formula, using sophisticated encoding techniques developed for planning. SATMC applies SAT solver to solve the propositional formula. The encoding techniques specifically used by SATMC are *linear encoding* and *graph-plan based encoding.* SATMC is the best choice to analyze web-browser based protocols, which normally send and receive messages over secure channels like Secure Socket Layer (SSL)/ Transport Layer Security (TLS). SATMC can also be used to validate complex temporal characteristics like fair exchange in a given protocol.

4.1.3 Constraint-Logic-based Attack Searcher (CL-AtSe)

CL-AtSe [83] takes input of a protocol specification in IF format, produced by the AVISPA compiler. CL-AtSe first applies rewriting technique on a given protocol specification to construct state transition diagram for a given protocol. Later it applies constraint solving technique to determine all reachable states for each participant in a given protocol and also decides the possibility of attack with respect to the Dolev-Yao intruder model. CL-AtSe was designed mainly to achieve *modularity* and *performance.* The modularity feature of CL-AtSe helps in extending and applying it for analyzing different classes of protocols. The performance feature in Ct-AtSe helps in obtaining the results quickly, specifically when protocol is analyzed with large number of sessions. The CL-Atse uses generic knowledge deduction rules from casrul [27] but with optimizations and extensions to analyze protocols. In order to minimize the search space CL-AtSe applies lazy intruder technique without eliminating any attack at the same time without adding new attacks. CL-AtSe integrates an optimized version of the Baader & Schulz unification algorithm [12], with modules for xor, exponentiation, and associative pairing. To improve the performance, CL-AtSe takes the advantage of optimization techniques like re-writing of input specification, simplification etc. CL-AtSe tries to combine as many steps as possible or allow them to execute as soon or as late, as possible. By doing so it can simplify the protocol in terms of the number of steps that are allowed to execute. The steps marked *as soon* are allowed to run immediately after it's parent step.

These marks will apply very restrictive constraints on interleaving step, which inturn greatly reduce CL-AtSe computing time. However, CL-AtSe can take such decisions, if it can build a proof automatically regarding the insecurity of a protocol. CL-Atse builds various protocol-dependent objects like a set of unforgeable terms like atoms, keys, etc. that the intruder cannot create. CL-Atse first tests the elements in a given step, then checks for the possibility of either merging or marking the step. Protocol optimization aim is to rewrite few parts of the protocol in order to improve the searching of an attack. The idea behind this one is to collect all possible origins from which ciphertexts can be send by intruder. Once exhaustive list of origins is built then, CL-Atse can make use of this list for future analysis.

4.1.4 Tree Automata based on Automatic Approximations for the Analysis of Security Protocols (TA4SP)

In order to validate the given security protocol, TA4SP [20], follows an approximation technique. TA4SP uses regular tree languages and rewriting to approximate the intruder knowledge. The term rewriting system represents not only the intruder abilities to analyse or compose the messages, but also the protocol steps. The main idea behind TA4SP is an extension of an approximation method based on tree automata. The approximation function used in TA4SP contains symbolic transitions (where the right hand side or the left hand side of a transition may contain variables). It is then possible to compute under-approximations, over-approximations and to switch automatically from the first to the second during the process. TA4SP may exhibit a flaw in a given security protocol by under-approximation or safe over any number of sessions by over-approximation. The AVISPA tool suits well to validate the properties of a security protocols with bounded number of sessions. The back-end analyzers exhibits security breaches if any exists in terms-of sequence of events that leads to violation of security goal. In next section, we discuss DNSSEC protocol specification in general notation.

4.2 An Overview of DNSSEC Specification Process

Before writing the actual DNSSEC protocol specification using HLPSL language 'to validate with the AVISPA tool', it is always suggested to represent the protocol using Alice - Bob (A - B) notation. The representation of a protocol using (A - B) will help us in getting clarity and better understanding of the protocol in terms of representation messages exchanged between principals. The DNSSEC protocol representation using (A - B) notion is as shown below.

```
Alice - Bob (A - B) Notation

1.  C  ->  S:  DNS_req
2.  S  ->  C:  DNS_res, RRSIG, DNSKEY(S)
3. C "verify" if  H(DNS_res) == {RRSIG}_DNSKY then
       go to Step 4.
       else discard DNS_res and exit.
4.  C  ->  R:  DS_req
5.  R  ->  C:  DS_res, RRSIG, DNSKEY(R)
6. C "verify" if  H(DS_res) == {RRSIG}_DNSKEY then
       go to step 7.
```

```
        else discard DS_res and exit
7. C "verify" if H(DNSKEY(S)) == (DS_res) then
   accept DNS_res.
   else discard DNS_res and exit.
```

In the above representation, we have used the following notation - Client(C), Server(S), Root(R). In step 1, client sends DNS request(DNS_{req}) to the server. In step 2, after receiving DNS request from the client, (here we are not showing the receiving of the message, which we are assuming it as implicit, because server will send a response only after receiving a request from client), server sends DNS response (DNS_{res}) signature of the response (RRSIG) and the public key of the server (DNSKEY) to client. In step 3, after receiving these three (DNS_{res}, RRSIG, and DNSKEY(S)) information, client verifies the integrity of DNS_{res}(i.e checking whether the DNS response gets modified during transit or not). Here, the integrity checking is carried out by computing hash value for DNS_{res} and is compared with the decrypted RRSIG value using public key of server (DNSKEY(S)). If both of these match, then client believes that the DNS_{res} did not get modified in transit and therefore the client accepts the DNS_{res} and continues with step 4. On the other hand if the DNS_{res} gets modified in transit, the client discards the DNS_{res} and exits the process.

In step 4, client sends the DS request (DS_{req}) to root server. In step 5, after receiving DS_{req} from client, server at root sends DS response (DS_{res}), signature of the response (RRSIG), and public key (DNSKEY) of root server to client. In step 6, after receiving these three (DS_{res}, RRSIG, and DNSKEY(R)) information, client verifies the integrity of DS_{res}. Here the integrity checking is carried out by computing the hash value for DS_{res} and compared with decrypted RRSIG value using public key (DNSKEY(R)). If both of them match then client believes that the DS_{res} did not get modified in transit and therefore the client accepts the DNS_{res} and continues to step 7. If this is not the case, the DS_{res} gets modified in transit and therefore the client discards the DS_{res} and exits the process. In step 7, we verify the public key sent by the server (DNSKEY(S)) and the hash value of the public key of server(DS_{res}) sent by the root. If both of them match, then client accepts the DNS_{res} sent by the server else client discards the DNS_{res} and exits the process. In steps 3 & 6 we verified the *data integrity* property and in step 7 we verified *data origin authentication* property of DNSSEC protocol. Before writing DNSSEC specification in HLPSL language, the general structure of HLPSL specification as well as the basic building blocks and predefined functions needs to be discussed.

4.3 AVISPA and HLPSL

In order to validate any security protocol using AVISPA tool, we need to write the specification of that protocol using the High Level Protocol Specification Language **(HLPSL)**. HLPSL is a simple and very convenient high level language to specify any security protocol for verification using automatic tools like AVISPA. Therefore in this section we discuss the major constructs of HLPSL that help in representing a protocol using HLPSL language as well as understanding any HLPSL specification. First we start with the overall structure of HLPSL specification.

`Structure of a HLPSL specification.`

$$\boxed{\begin{array}{c} BasicRole\_definition+ \\ CompositionRole\_definition? \\ EnvironmentRole\_definition? \\ Goal\_declaration? \end{array}}$$

HLPSL is basically a role based language, which means each participant in a security protocol is represented as one role in HLPSL specification, including the environment in which the protocol is going to execute. In general, any HLPSL specification of a protocol consists of one or more basic role definitions, followed by the definition of composite role where we combine one or more basic roles together so that they get executed together, usually in parallel. Environment role is defined wherein we instantiate composition role to create one or more sessions. Finally, we specify security parameters as goals that we want to validate by the tool. The basic roles are played by agents, and the format of the basic role is as shown below.

`Basic role structure`

$$\boxed{\begin{array}{c} \textit{role role\_name(list of parameters)} \\ \textit{played\_by name\_of\_player def} = \\ \textit{local variables declaration section.} \\ \textit{init section.} \\ \textit{transitions declaration section.} \\ \textit{end role} \end{array}}$$

The basic role definition consists of list of parameters, local variables, `init` section and transition section. The role definition starts from key word def and ends with keywords end role. List of parameters are used to send and receive messages from one role to another. Local variable section declares local variables that are used within the role. Init section is used to initialize the variables. Transition section is used to specify the actions like sending, receiving, encrypting, decrypting of messages, etc that are performed by the agent of a role.

`Transition structure`

$$\boxed{\begin{array}{c} \textit{Label}\ ".\text{"}\ \textit{Predicates}\ "--|>"\ \textit{Actions}\ \%\ \textit{spontaneous action} \\ |\ \textit{Label}\ ".\text{"}\ \textit{Events}\ "=|>"\ \textit{Reactions}\ \%\ \textit{immediate reaction} \\ \textit{Label}\ ::=\ \textit{const\_ident}\ |\ \textit{nat\_ident} \end{array}}$$

The transitions in a basic role are of two types: one is spontaneous actions, that gets executed when the state predicates on the left-hand side are satisfied, and the other one is immediate reactions, that gets executed immediately whenever the non-stutter events (that is events based on the change of some variables values) on the left-hand side are satisfied. The transaction format is as shown below.

`Composition role structure`

$"role"\ role\_name(list\ of\ parameters)$
$"def="$
local variables declaration section.
init section.
composition declaration section.
$"end"\ "role"$

The composition role definition consists of list of parameters, local variables, init section and composition section. The role definition starts from key word def and ends with keywords end role. List of parameters are used to communicate messages from one role to another. Local variables section is used to declare local variables that are used within the role. Init section is used to initialize the variables. Composition section is used to combine one or more basic roles so that they can execute in parallel.

`Environment role structure`

$"role"environment()$
$"def="$*const declaration section.*
intruder knowledge declaration section.
compositon section.
$"end"\ "role"$

The environment role is a starting point of the execution of a program. The role definition starts from the key word def and ends with keywords end role. The environment role consists of a constant declaration section, intruder knowledge declaration section and composition section. In constant declaration section we declare the constants of agent type, key type, protocol id type, etc. Knowledge of an intruder is composed of assigned variables, constants and also the knowledge given to all the instances of basic and composition roles. In the environment section, we instantiate one or more composition role which was declared earlier.

The goals of a security protocol can be declared as any of the following type.
1. Some information can be declared as secret so that privacy is maintained.
2. Agents can declare strong authentication requirement on some information.
3. Agents can declare weak authentication requirement on some information.
To declare the above goals HLPSL provides predefined macros. `secret(E,id,S):` information E is declared as secret and it is shared by a set of agents represented as S; secret information E is identified in goal section as *id*. `witness(A,B,id,E):` agent A authenticates (weak) information E, that was sent to agent B; in goal section this can be identified as *id*. `request(B,A,id,E):` agent B requests a check for strong authentication on the value E by agent A; in goal section this can be identified as *id*. `wrequest(B,A,id,E):` it is similar to that of a request, but this is used for a weak authentication property.

HLPSL supports some basic building blocks which will help the protocol specifiers to quickly write the specification of the protocol in HLPSL for verification. Basic types available for specifying protocols can be described as follows: *agent, channel, boolean, integer, text, message, public_key, symmetric_key etc.. Aggregate types: tuples, sets, lists..* Variables can be assigned with fresh values (using *new()*). Functions : HLPSL supports functions like cryptographic hash functions, encryption and decryption. *E.g. H(x), x_Ka, x_inv(ka) etc..* AVISPA uses Dolev-Yao (DY) channels where the presence of intruder is also considered.

4.4 Validating DNSSEC using AVISPA

As it has been discussed earlier that each participant in the protocol is represented as one role in AVISPA, here we define one role for client, one role for server and one role for root. After defining the basic roles, we defined the composition role to combine the client, server and root together so as to execute in a parallel fashion. After that we define the environment role to start the session between the roles combined in composition role and also to start the execution of the program. Finally, we define the goals of the protocol that we want to validate using AVISPA tool.

Simulation of DNSSEC Protocol

Role of Client

```
role client( C, S, R : agent,
% C, S, R are of type agent, play the roles of client,
server and root respectively.
          Kc, Kr, Ks : public_key,
        % Kc, Kr, Ks are the public keys of client, root,
        server respectively.
       DNS_req, DNS_res,
      DS_req,  DS_res : text,
      % DNS_req, DNS_res are DNS request and DNS response
      % respectively. DS_req, DS_res are DS request and DS
      % response respectively.
                  SND, RCV : channel(dy))
          % SND, RCV are send and receive channels of type
          Dolev Yao respectively.
     played_by C def=
   local
            State : nat,
                    H : hash_func,
               Ns, Nr : text,
            R1, R2 : text,
         RRSIG1, RRSIG : text,
         DNSKeyMap : (agent.public_key) set
   init
 State:=0 /\ DNSKeyMap:={(C.Kc),(R.Kr)}
   transition
% Client sends a DNS request to server
    1. State = 0 /\ RCV(start) =|>
```

```
     State':=1 /\ SND(S.DNS_req)
    % After receiving the DNS response, client computes the
    % hash value for the received response and compares with
    % the received hash value, if both are matched then Client
    % sends DS request to the root, here we are assuming that,
    % root is the parent of server.
    2. State = 1 /\ RCV(C.DNS_res) /\ RCV(C.Ks)
                 /\ RCV(C.{H(DNS_res)}_inv(Ks))
             =|> State':=2 /\ SND(R.DS_req)
             /\ request(C,S,c_s_ks,inv(Ks))
% Client receives DS resopnse from root, client computes
% the hash value for the received response and compares
% with the received hash value, if both are matched then
% client accepts the DNS response received from server
% otherwise it rejects.

    3. State = 2 /\ RCV(C.DS_res)  /\ RCV(C.Kr)
                 /\ RCV(C.{H(DS_res)}_inv(Kr))
               =|> State':=3  /\ request(C,R,c_r_kr,inv(Kr))
end role
```

Role of Server

```
role server(       C, S, R : agent,
                 Kr, Ks : public_key,
        DNS_req, DNS_res : text,
                SND, RCV : channel(dy))
     played_by S def=
local
       State : nat,
           H : hash_func,
       RRSIG : text,
          Ns : text,
       DNSKeyMap : (agent.public_key) set,
        DSKeyMap : (agent.text) set,
     DomNamIPMap : (text.text) set


init
State:=0 /\ DNSKeyMap:={(S.Ks),(R.Kr)}
transition
% Server receives the DNS request from client
% and sends the DNS respone as well as the
% signature of the response to the client
1. State = 0 /\ RCV(S.DNS_req) =|> State':=1
/\ SND(C.DNS_res)  /\ SND(C.{H(DNS_res)}_inv(Ks))
/\ SND(C.Ks) /\ witness(S,C,c_s_ks,inv(Ks))
end role
```

Role of Root Server

```
role root(            C, R : agent,
               Kr : public_key,
         DS_req, DS_res : text,
               SND, RCV : channel(dy))
     played_by R def=
local
       State : nat,
          RRSIG1 : text,
      DomainName : text,
       IPAddress : text,
          Nr : text,
           H : hash_func,
       DNSKeyMap : (agent.public_key) set,
        DSKeyMap : (agent.text) set,
     DomNamIPMap : (text.text) set


      init
State:=0 /\ DNSKeyMap:= {(R.Kr)}
transition
% Root receives the DS request from client and sends the
% DS respone as well as the signature of the response to
% the client
1. State = 0 /\ RCV(R.DS_req) =|> State':=1 /\ SND(C.DS_res)
  /\ SND(C.{H(DS_res)}_inv(Kr)) /\ SND(C.Kr)
  /\ witness(R,C,c_r_kr,inv(Kr))
end role
```

Composition Role

```
role session(    C, S, R : agent,
        Kc, Ks, Kr : public_key)
     def=
local
         CS,CR,SS,SR,RS,RR : channel(dy),
             DNs_req, DNS_res,
                   DS_req, DS_res,
                            RRSIG : text
composition
client( C, S, R, Kc, Kr, Ks, DNs_req, DNS_res, DS_req,
        DS_res, CS, CR) /\
server( C, S, R, Kr, Ks, DNs_req, DNS_res, SS, SR) /\
root( C, R, Kr, DS_req, DS_res, RS, RR)
end role
```

Role of Environment

```
%% Main role
```

```
role environment()
     def=
const
         c, s, r : agent,
          kc, kr, ks : public_key,
               h : hash_func,
          c_s_ks,
                 c_r_kr : protocol_id

intruder_knowledge  =  { c, s, r, h, kc, kr, ks }
composition
session(c, s, r, kc, ks, kr) /\ session(c, s, r, kc, ks, kr)
end role
```

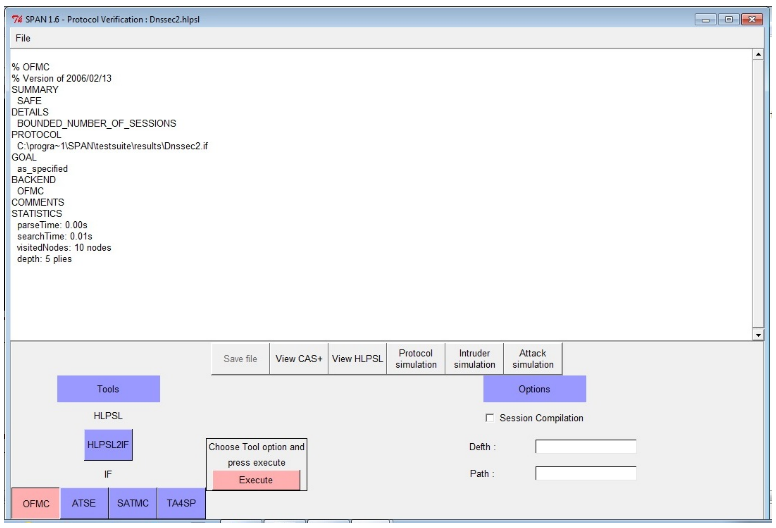
Goal section

```
goal


  authentication_on  c_r_kr
authentication_on  c_s_ks
end  goal
environment()
```

4.5 Results

The following results are obtained after the DNSSEC protocol specification is validated using AVISPA tool. Here we are displaying the results obtained from two back-end verifiers OFMC and CL-AtSe. First we display the results related to a single session among the client, server and the root with the intruder and without the intruder. Next we display the results related to two sessions executing in parallell.

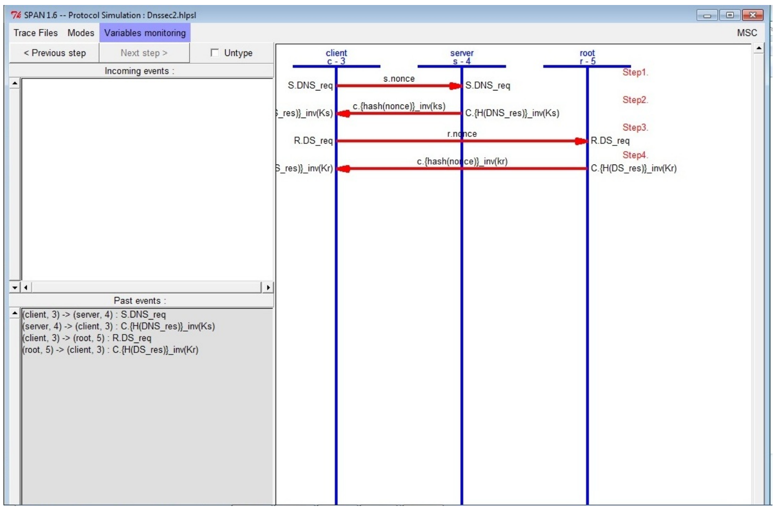
4.5.1 Results with On-the-fly Model-Checker (OFMC) as back-end



```
% OFMC
% Version of 2006/02/13
SUMMARY
  SAFE
DETAILS
  BOUNDED_NUMBER_OF_SESSIONS
PROTOCOL
  C:\progra~1\SPAN\testsuite\results\Dnssec2.if
GOAL
  as_specified
BACKEND
  OFMC
COMMENTS
STATISTICS
  parseTime: 0.00s
  searchTime: 0.01s
  visitedNodes: 10 nodes
  depth: 5 plies
```

Figure 4.2: OFMC Summary for DNSSEC Protocol with one session

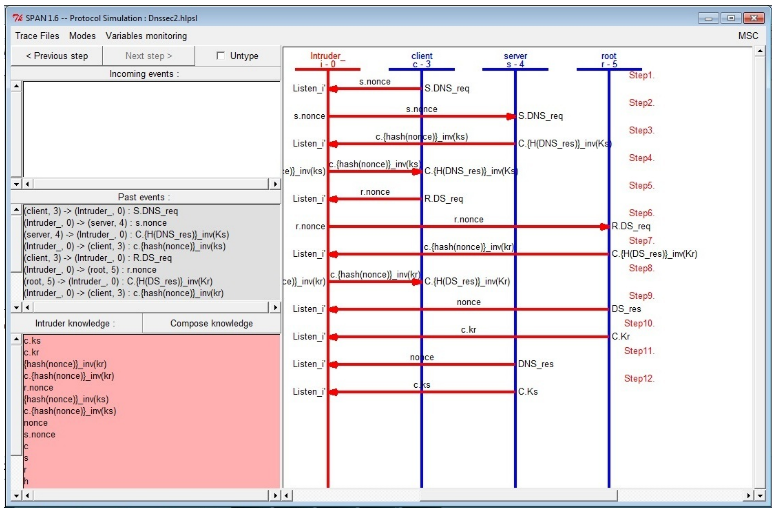
Figure 4.2 shows execution summary of OFMC. It is clear from Figure 4.2 that, the given specification is SAFE (it means no attack is possible under specified constraints) and specified goals are fulfilled. We can also see the execution statistics like parse time, search time, no of nodes visited and depth.



```
(client, 3) -> (server, 4) : S.DNS_req
(server, 4) -> (client, 3) : C.{H(DNS_res)}_inv(Ks)
(client, 3) -> (root, 5) : R.DS_req
(root, 5) -> (client, 3) : C.{H(DS_res)}_inv(Kr)
```

Figure 4.3: OFMC simulation for DNSSEC Protocol with one session

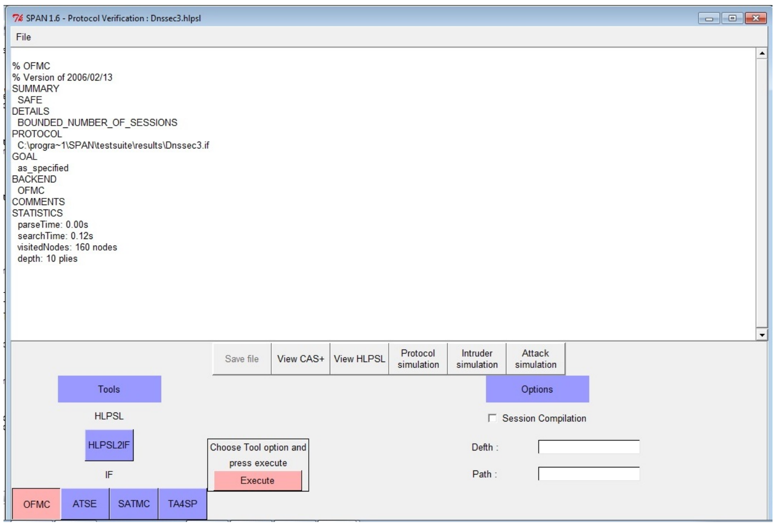
Figure 4.3 shows the interactions among the client, the server and the root server. First, client sends DNS request to server and then server sends a response. In order to authenticate the response, client sends DS request to the root server. Root server sends the DS response and client can make use of this response to authenticate the DNS response from the server. Figure 4.4 is similar to Figure 4.3, except communication between the client and the server as well as the client and the root server will pass through the intruder. Result shows that even in the presence of intruder, communication is not getting disturbed as the results got are the same.



```
(client, 3) -> (Intruder_, 0) : S.DNS_req
(Intruder_, 0) -> (server, 4) : s.nonce
(server, 4) -> (Intruder_, 0) : C.{H(DNS_res)}_inv(Ks)
(Intruder_, 0) -> (client, 3) : c.{hash(nonce)}_inv(ks)
(client, 3) -> (Intruder_, 0) : R.DS_req
(Intruder_, 0) -> (root, 5) : r.nonce
(root, 5) -> (Intruder_, 0) : C.{H(DS_res)}_inv(Kr)
(Intruder_, 0) -> (client, 3) : c.{hash(nonce)}_inv(kr)
```

```
c.ks
c.kr
{hash(nonce)}_inv(kr)
c.{hash(nonce)}_inv(kr)
r.nonce
{hash(nonce)}_inv(ks)
c.{hash(nonce)}_inv(ks)
nonce
s.nonce
c
s
r
h
```

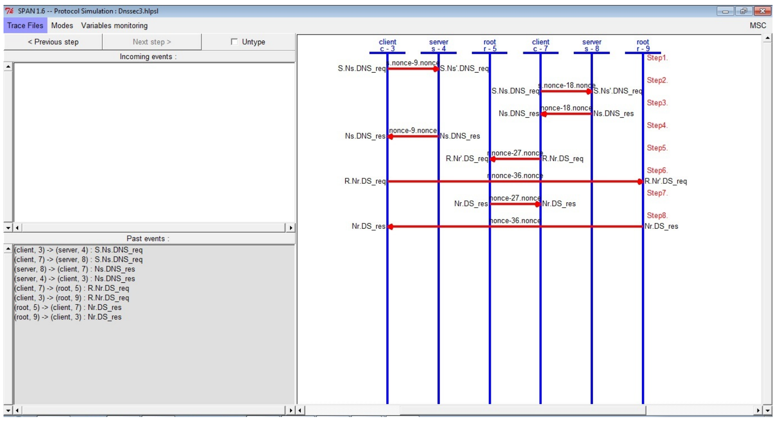
Figure 4.4: OFMC simulation for DNSSEC Protocol with one session including intruder



```
% OFMC
% Version of 2006/02/13
SUMMARY
  SAFE
DETAILS
  BOUNDED_NUMBER_OF_SESSIONS
PROTOCOL
  C:\progra~1\SPAN\testsuite\results\Dnssec3.if
GOAL
  as_specified
BACKEND
  OFMC
COMMENTS
STATISTICS
  parseTime: 0.00s
  searchTime: 0.12s
  visitedNodes: 160 nodes
  depth: 10 plies
```

Figure 4.5: OFMC Summary for DNSSEC Protocol with two sessions

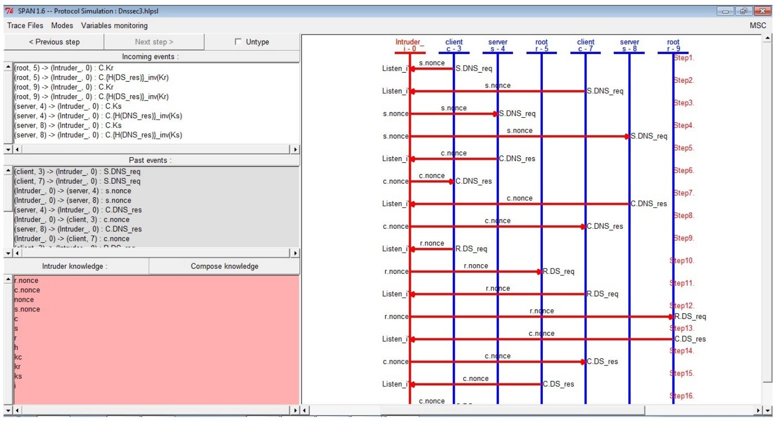
Figure 4.5 shows summary of execution of a DNSSEC protocol with two sessions. It is clear from Figure 4.5 that, the given specification is SAFE (it means no attack is possible under specified constraints) and specified goals are fulfilled. We can also see the execution statistics like parse time, search time, number of nodes visited and depth.



```
(client, 3) -> (server, 4) : S.Ns.DNS_req
(client, 7) -> (server, 8) : S.Ns.DNS_req
(server, 8) -> (client, 7) : Ns.DNS_res
(server, 4) -> (client, 3) : Ns.DNS_res
(client, 7) -> (root, 5) : R.Nr.DS_req
(client, 3) -> (root, 9) : R.Nr.DS_req
(root, 5) -> (client, 7) : Nr.DS_res
(root, 9) -> (client, 3) : Nr.DS_res
```

Figure 4.6: OFMC simulation for DNSSEC Protocol with two sessions

Figure 4.6 shows the interactions among client, server and root server. It is the same as that of Figure 4.3, but with two parallel sessions. Figure 4.7 is similar to that of Figure 4.6 except for the fact that the intruder is taken into consideration.



Incoming events :

```
(root, 5) -> (Intruder_, 0) : C.Kr
(root, 5) -> (Intruder_, 0) : C.{H(DS_res)}_inv(Kr)
(root, 9) -> (Intruder_, 0) : C.Kr
(root, 9) -> (Intruder_, 0) : C.{H(DS_res)}_inv(Kr)
(server, 4) -> (Intruder_, 0) : C.Ks
(server, 4) -> (Intruder_, 0) : C.{H(DNS_res)}_inv(Ks)
(server, 8) -> (Intruder_, 0) : C.Ks
(server, 8) -> (Intruder_, 0) : C.{H(DNS_res)}_inv(Ks)
```

Past events :

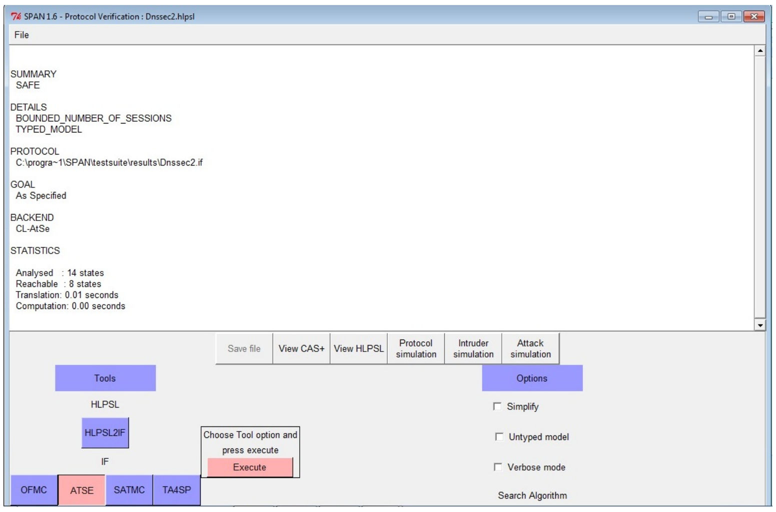
```
(client, 3) -> (Intruder_, 0) : S.DNS_req
(client, 7) -> (Intruder_, 0) : S.DNS_req
(Intruder_, 0) -> (server, 4) : s.nonce
(Intruder_, 0) -> (server, 8) : s.nonce
(server, 4) -> (Intruder_, 0) : C.DNS_res
(Intruder_, 0) -> (client, 3) : c.nonce
(server, 8) -> (Intruder_, 0) : C.DNS_res
(Intruder_, 0) -> (client, 7) : c.nonce
```

Intruder knowledge :

```
r.nonce
c.nonce
nonce
s.nonce
c
s
r
h
kc
kr
ks
i
```

Figure 4.7: OFMC simulation for DNSSEC Protocol with two sessions including intruder

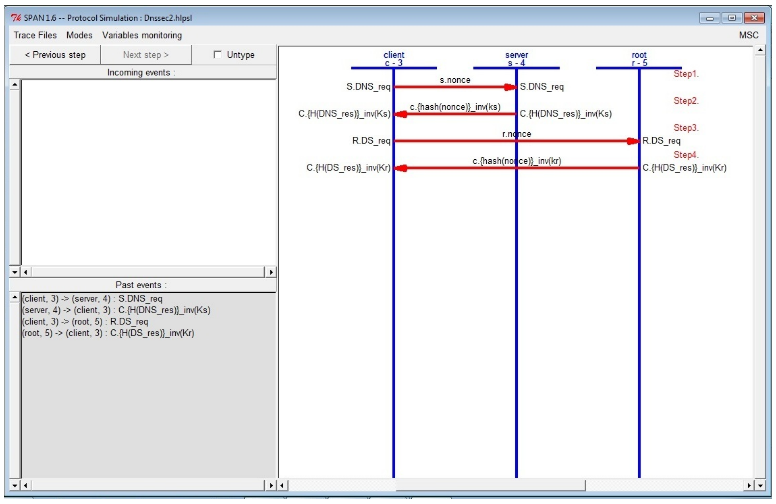
4.5.2 Results using Constraint-Logic-based Attack Searcher (CL-AtSe) as a back-end



```
SUMMARY
 SAFE

DETAILS
 BOUNDED_NUMBER_OF_SESSIONS
 TYPED_MODEL

PROTOCOL
 C:\progra~1\SPAN\testsuite\results\Dnssec2.if

GOAL
 As Specified

BACKEND
 CL-AtSe

STATISTICS

 Analysed   : 14 states
 Reachable  : 8 states
 Translation: 0.01 seconds
 Computation: 0.00 seconds
```

Figure 4.8: CL-AtSe Summary for DNSSEC Protocol with one session

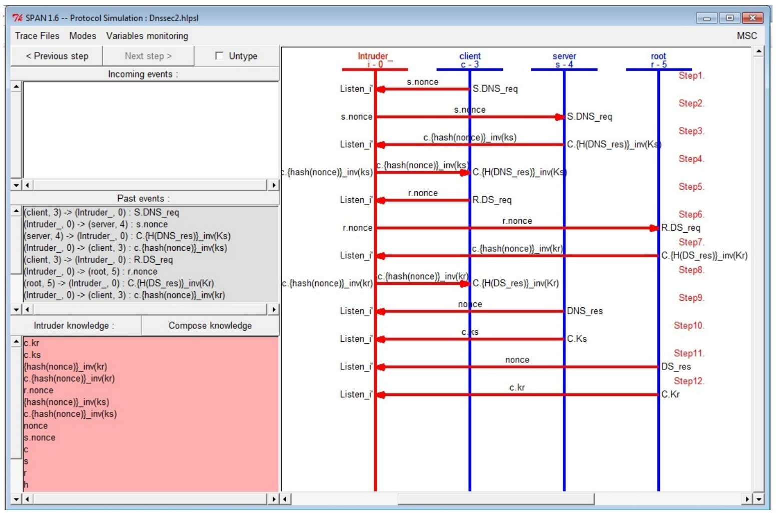
Figure 4.8 shows execution summary of CL-TtSe. It is clear from Figure 4.8 that, the given specification is SAFE (it means no attack is possible under specified constraints) and specified goals are fulfilled. We can also see the execution statistics like parse time, search time, no of nodes visited and depth.



```
(client, 3) -> (server, 4) : S.DNS_req
(server, 4) -> (client, 3) : C.{H(DNS_res)}_inv(Ks)
(client, 3) -> (root, 5) : R.DS_req
(root, 5) -> (client, 3) : C.{H(DS_res)}_inv(Kr)
```

Figure 4.9: CL-AtSe simulation for DNSSEC Protocol with One session

Figure 4.9 shows interactions among client, server and root server. First, client sends DNS request to server, then the server sends response. In order to authenticate the response, client sends DS request to root server. Root server sends DS response and client makes use of this response to authenticate DNS response from server.

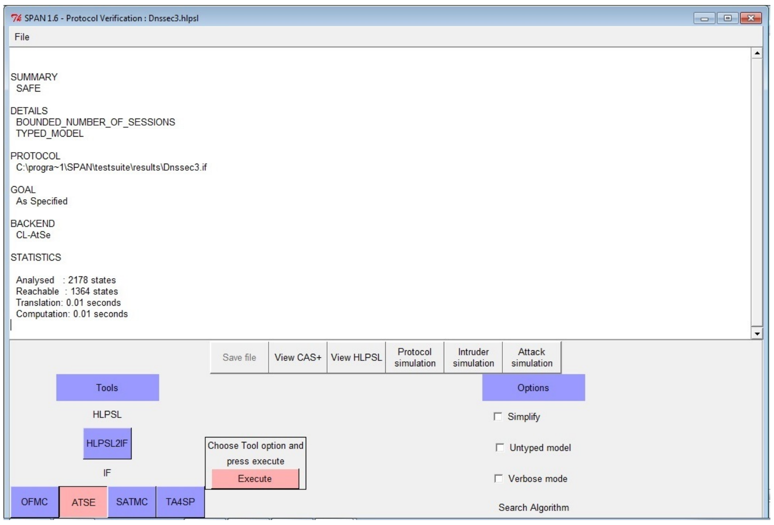


```
(client, 3) -> (Intruder_, 0) : S.DNS_req
(Intruder_, 0) -> (server, 4) : s.nonce
(server, 4) -> (Intruder_, 0) : C.{H(DNS_res)}_inv(Ks)
(Intruder_, 0) -> (client, 3) : c.{hash(nonce)}_inv(ks)
(client, 3) -> (Intruder_, 0) : R.DS_req
(Intruder_, 0) -> (root, 5) : r.nonce
(root, 5) -> (Intruder_, 0) : C.{H(DS_res)}_inv(Kr)
(Intruder_, 0) -> (client, 3) : c.{hash(nonce)}_inv(kr)
```

```
c.kr
c.ks
{hash(nonce)}_inv(kr)
c.{hash(nonce)}_inv(kr)
r.nonce
{hash(nonce)}_inv(ks)
c.{hash(nonce)}_inv(ks)
nonce
s.nonce
c
s
r
h
```

Figure 4.10: CL-AtSe simulation for DNSSEC Protocol with one session including intruder

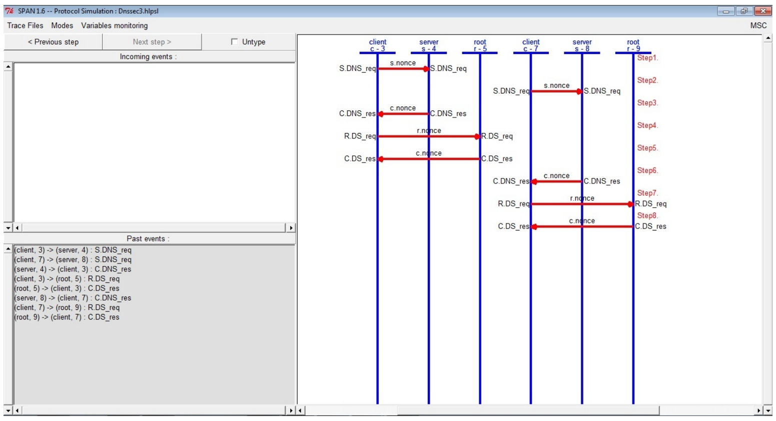
Figure 4.10 is the same as Figure 4.9, except that communication between client and server as well as client and root server will pass through the intruder. It shows that even in the presence of intruder, communication did not get disturbed as the results got are the same.



```
SUMMARY
  SAFE

DETAILS
  BOUNDED_NUMBER_OF_SESSIONS
  TYPED_MODEL

PROTOCOL
  C:\progra~1\SPAN\testsuite\results\Dnssec3.if

GOAL
  As Specified

BACKEND
  CL-AtSe

STATISTICS

  Analysed   : 2178 states
  Reachable  : 1364 states
  Translation: 0.01 seconds
  Computation: 0.01 seconds
```

Figure 4.11: CL-AtSe Summary for DNSSEC Protocol with two sessions

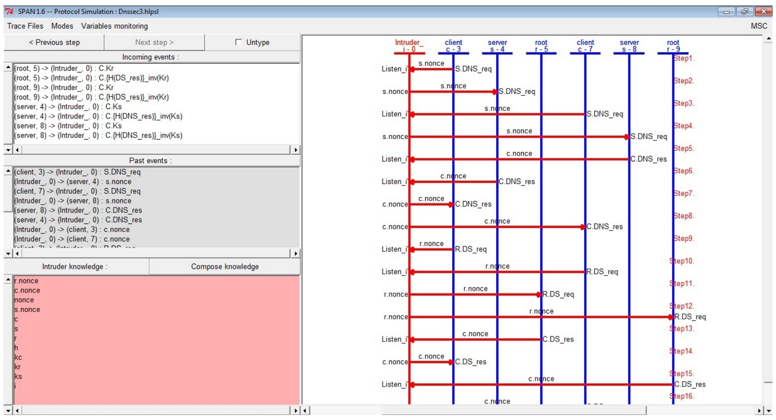
Figure 4.11 shows summary of execution for DNSSEC protocol with two sessions. It is clear from Figure 4.11 that, the given specification is SAFE (it means no attack is possible under specified constraints) and specified goals are fulfilled. Also shown is the execution statistics like parse time, search time, no of nodes visited and depth.



```
(client, 3) -> (server, 4) : S.DNS_req
(client, 7) -> (server, 8) : S.DNS_req
(server, 4) -> (client, 3) : C.DNS_res
(client, 3) -> (root, 5) : R.DS_req
(root, 5) -> (client, 3) : C.DS_res
(server, 8) -> (client, 7) : C.DNS_res
(client, 7) -> (root, 9) : R.DS_req
(root, 9) -> (client, 7) : C.DS_res
```

Figure 4.12: CL-AtSe simulation for DNSSEC Protocol with two sessions

Figure 4.12 shows iterations between client, server and root server. It is same as Figure 4.9, but with two parallel sessions. Figure 4.13 is similar to Figure 4.12, except that the intruder is taken into consideration.



```
Incoming events :
(root, 5) -> (Intruder_, 0) : C.Kr
(root, 5) -> (Intruder_, 0) : C.{H(DS_res)}_inv(Kr)
(root, 9) -> (Intruder_, 0) : C.Kr
(root, 9) -> (Intruder_, 0) : C.{H(DS_res)}_inv(Kr)
(server, 4) -> (Intruder_, 0) : C.Ks
(server, 4) -> (Intruder_, 0) : C.{H(DNS_res)}_inv(Ks)
(server, 8) -> (Intruder_, 0) : C.Ks
(server, 8) -> (Intruder_, 0) : C.{H(DNS_res)}_inv(Ks)

Past events :
(client, 3) -> (Intruder_, 0) : S.DNS_req
(Intruder_, 0) -> (server, 4) : s.nonce
(client, 7) -> (Intruder_, 0) : S.DNS_req
(Intruder_, 0) -> (server, 8) : s.nonce
(server, 8) -> (Intruder_, 0) : C.DNS_res
(server, 4) -> (Intruder_, 0) : C.DNS_res
(Intruder_, 0) -> (client, 3) : c.nonce
(Intruder_, 0) -> (client, 7) : c.nonce

Intruder knowledge :
r.nonce
c.nonce
nonce
s.nonce
c
s
r
h
kc
kr
ks
i
```

Figure 4.13: CL-AtSe simulation for DNSSEC Protocol with two sessions including intruder

4.6 Summary

In this chapter we demonstrated how the AVISPA tool can be used to validate the DNSSEC protocol. We have represented DNSSEC protocol in HLPSL which is later translated into IF Form by HLPSLIF converter. The IF form of the protocol is, then analyzed using four backend analyzers called OFMC, SATMC, CL-AtSe and TA4SP to check the correctness of the given protocol. The simulation results ensured that the DNSSEC protocol is safe which means that no attacks are possible on specified goals. We have also displayed the results pertaining to single as well as multiple sessions, with different backend analysers. Finally we concluded that the DNSSEC protocol is successfully validated as no attacks found against it's *authentication* and *integrity* goals.

Chapter 5

Analysing DNSSEC as a Planning problem in Multi-Agent Systems

In the previous two chapters we discussed about the formal analysis and verification of DNSSEC in the background of SVO logic and provided validation of DNSSEC using the AVISPA tool. In this chapter we deviate a bit and try to address the analysis and verification of DNSSEC from a planning problem perspective in a Multi Agent System environment.[1]

A planning problem in Artificial Intelligence (AI) is a `representation` in some formal language which includes the following three steps: a) description of the initial state of the world/environment b) description of the goal state to be achieved and c) description of the possible actions that can be performed. In this setup, a possible solution to a `problem` in a planning domain would be a *plan* consisting of a sequence of actions which when executed would satisfy the initial descriptions of the environment and achieves the goal. When it comes to security protocols, a planning problem can be visualised in the background of multi-agent systems (MAS) [90] wherein the agents exchange messages and are

[1]Parts of this chapter has been published/accepted as
1. *Incorporating temporal planning within a BDI architecture*, 5th Indian International Conference on Artificial Intelligence (IICAI 2011) pages, 1618 - 1636.
2. *BDI based performance enhancement in DNSSEC*, Accepted in the 14th International conference on Distributed Computing and Internet technology (ICDCIT 2018), 11th-13th January 2018.

subject to attacks by the intruders. Though these agents can cooperate, coordinate and negotiate to achieve their goals, as far as secure communication in MAS is concerned, the problem still remains as to whether a message sent by an agent is reliably received by other agents and whether the message received is regarded as reliable in the view of receivers. As pointed out in [71], the problem generally depends on the *trust* that agents would put in the security mechanisms of a system .

Several modal logics such as belief logics and logics of knowledge [84] have been proposed in the literature for the specification and verification of different aspects of multi agent systems including that of *security* and *trust*. One of the key aspects that should be taken care of while designing security protocols using logic for agent based system is with respect to modelling and reasoning about time. Modelling time explicitly is important especially for security protocols using *timestamps* because one can add checks on the expiration dates of the certificates. In the case of DNSSEC there is an explicit start time and end time for its signatures. It is extremely important to set the validating resolver's internal clock to the correct time because if wrong time is set then the resolver can interpret valid signatures as expired and vice-versa. Many works have been proposed in the literature for analysing authentication protocols in Agent-Based Systems wherein an explicit notion of time is introduced using variants of temporal logic [56].

Our approach in this Chapter is not to provide an agent based formalism based on temporal logic for analysing the DNSSEC protocol. We would like to devise a declarative specification of the world wherein the proof of the existence of a goal state can be linked to a plan by visualising the plan as a sequence of steps to achieve the goal. The planning problem then becomes the problem of finding a time t such that the goal g holds wherein the goal could be a conjunction of formulae along with a specific set of actions. On the other hand, if it is not possible to find a model for a goal state then one can prove that there is no plan that achieves the goal. In a way our approach is similar to using logic programs to formalise protocols. In order to achieve this end we have chosen the Beleif, Desire, Intention (BDI) framework as our underlying agent model. Initially we demonstrate how to incorporate the temporal aspect into the BDI architecture. Thereafter, in the next Chapter we show how a variant of such a model can be used in a security protocol like DNSSEC.

5.1 Introduction to BDI

BDI (Belief, Desire, Intention) is a software model designed for programming intelligent agents and is based on the theory of practical reasoning put forward by Michael E. Bratman [22] in 1987. According to Bratman the concepts of Beliefs, Desires and Intentions should be represented and reasoned in a way such that it can lead to an explanation for our actions. Beliefs form the dataset of things that we are aware of as far as the environment is concerned whereas desires and intentions are pro-attitutudes that can lead to actions. Bratman considers intention as slightly different from desires as he terms intention as conduct-controlling-pro-attitude having an element of commitment attached to it. Bratman also talks in depth in his book of the role of plans in human practical reasoning wherein he describes about the *temporal persistence* in plans as well as the formation of future plans being depended on prior committed plans. Any system/software that can be characterised using these notions of Belief, Desire, Intention and Plans had the acronym `Intelligent Agents` and these concepts were used in a very loose sense to solve problems in the domain of agent programming.

Many logical models were proposed to define and reason about BDI agents which led to the development of different axiomatisations like BDICTL [35, 72], LORA [90], BOID [23] etc. Along with the logical models there are also numerous implementation of the BDI model like AgentSpeak-L, JAM, JACK, JASON, JADEX, SPARK, 3APL, 2APL, GOAL, CogniTao, Living Systems Process Suite etc. The success of the `BDI` [44] software model can be attributed to its ability for imparting reasoning in resource bounded agents which have limited memory and computing power as well as its application in dynamic environments. This ability stems from the fact that the BDI paradigm provides a mechanism that differentiates *plan selection* (choosing what to do) from *plan execution* (doing it) i.e., a balance is maintained on the `time` spent on deliberating about plans and executing those plans.

One of the most important activity involved in the BDI software model is the generation of a plan for a given objective i.e., finding a sequence of actions which can fulfil the given objectives or intentions. In most of the BDI agent systems this has been implemented as a plan-library, where a predefined set of plans is incorporated for every possible goal. Once plan-library is implemented then the planning problem can be reduced to a mapping whereby an agent can select appropriate plans from the plan-library for a given goal. A crucial aspect that has not been given much importance in the BDI planning domain is with respect to the *creation* of plans in the first place. Is there any justification for creating a plan as such so that it can be incorporated in the plan library. Usually this part is left to the system designer/programmer. Secondly *temporal persistence* of a plan in the sense of explicit reference to time, is not explored. In the BDI model usually plans have a hierarchical nature in which one plan may consist of a number of steps. It is argued that the overall plan remains in effect while the subsidiary plans are being executed and this in a way captures the temporal persistence. In this Chapter we make two important contributions.

- We extend the basic BDI architecture so as to include an action-base with temporal durations.
- Extend the existing classical planning cycle in BDI with temporal planning using our proposed `GeneratePlan` method.

We extend the basic BDI architecture and the classical planning life cycle, to incorporate temporal planning. The extended BDI architecture provides a provision of generating a plan for a given goal, when a classical planning cycle fails to do so.

5.2 BDI Architecture

As mentioned in the previous section the **BDI** architecture [21] is based on the paradigm of belief-desire-intention that represent the informational, motivational and decision-making ability of an agent. The constructs of belief, desire and intention can be interpreted as follows: **Beliefs** represent the facts about a given world at a given time, **Desires** represent goals or some desired end states and **Intentions** represent an agent's commitment to its desires (goals) and to the plans that will lead to achieve those goals.

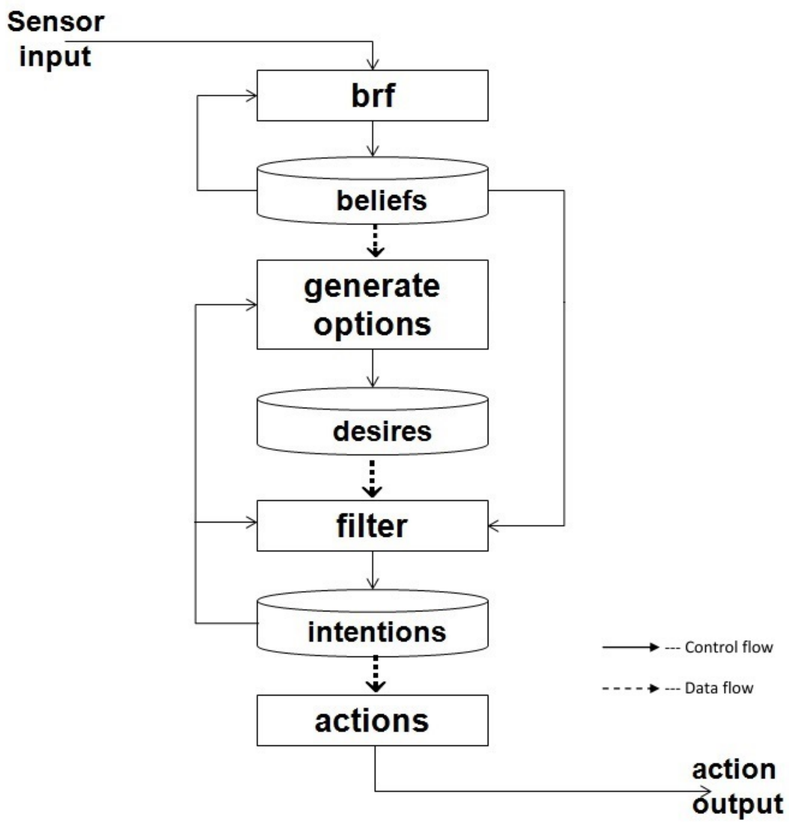


Figure 5.1: BDI Architecture.

In the BDI architecture shown in Figure 5.1, the agent first receives data from the sensors about its environment. After processing the data collected from the sensors, the agent stores the data as beliefs or facts in a storage called belief-base. In other cases, agent's beliefs may be based on internal inferences or derivations based on the existing beliefs. For example, for a given data of birthday, we may infer the age as per date. The belief-base is considered as an informative component of the system. The beliefs stored in the belief-base are true only at a given point of time or for a given period of time. The beliefs in a belief-base get updated by a function called *belief revision function* (brf) which takes a perceptual input or current beliefs as input and then derives a new set of beliefs as output. The option generation function takes current beliefs and current intentions as the input and determines the options available to the agent. That means option generation function updates the desires in desire-base based on current beliefs and intentions. Option generation function ensures that the goals (which are stored as desires in the desire-base) are concrete and consistent. For example, we can't store *stay at home* and *go to movie* as desires simultaneously, because these two goals are mutually exclusive. The desire base stores the information of the goals to be achieved, as well as properties and costs associated with each goal. The desire-base is considered as motivational state of the system. The filter function (filter) acts as a deliberation function and determines the agent's intentions on the basis of its current beliefs, desires, and already formed intentions. A set of intentions that are stored in the *intention-base* represent the committed goals that are in focus. In order to implement the committed goals, the action selection function selects and sequences a set of actions which can also be seen as the preparation of a plan. Finally, the agent executes the plan to achieve the current intentions.

5.3 Classical Planning Cycle

The planning problem in a BDI system can be broadly classified into two types: 1) domain independent planning [36] (also called classical planning [34]/first principles planning) and 2) domain dependent plannin [77]. In classical planning, actions and planning problem specifications are used to generate a plan and thereafter the plan is applied to the initial state of the system and modifications are made such that finally the goal state is eached. Domain dependent planning takes domain control knowledge as additional input to specify the those actions that should be considered and the order in which they should be used at different states of a planning process. The plan generation is faster in domain dependent planning than domain independent planning but the control knowledge used in domain dependent planning may also restrict the space of possible plans. Here we demonstrate one simple problem to show how a classical plan is generated as shown in Table 5.1. The language called STRIPS (Stanford Research Institute Problem Solver) is used to represent the planning problem.

> **Problem:** Replace the flat tyre (punctured tyre) with spare tyre.
> Initially The flat tyre is at axle and spare tyre is at truck.
> **Solution:**
> `Initial state`: At(flat tyre, axle), At(spare, truck)
> `Goal state`: At(flat tyre, truck), At(spare, axle)
> `Actions`: Remove(flat tyre, axle), Remove(spare, truck),
> Fix(spare, axle), Fix(flat tyre, truck)
> `Plan`:
> 1. Remove(flat tyre, axle)
> *precondition*: At(flat tyre, axle)
> *effect*: $\neg$ At(flat tyre, axle)
> 2. Remove(spare, truck)
> *precondition*: At(spare, truck)
> *effect*: $\neg$ At(spare, truck)
> 3. Fix(spare, axle)
> *precondition*:$\neg$ At(spare, truck)
> *effect*: At(spare, axle)
> 4. Fix(flat tyre, truck)
> *precondition*:$\neg$ At(flat tyre, axle)
> *effect*: At(flat tyre, truck)

Table 5.1: An example of classical planning

The characteristics a problem should possess in order to consider it as a classical planning problem are as follows: First, we should be able to represent the problem using finite number of states, actions and events. Second, it must be fully observable, that means, at any point of time we should be in a position to find the current state for a given input. Third, it must be a deterministic system, that means, for each action there should be only one outcome (next state). Fourth, it must be static, that is, no changes are allowed during execution. Fifth, the goal states must be clearly specified. Sixth, a plan generated to solve the given problem must be a linearly ordered sequence of actions. Seventh, it is assumed of implicit time, that is, all events occur instantaneously.

The actions in the above plan get executed only when the preconditions of that action get satisfied. The effect in the action represents the result obtained after completion of the action. The plan can be generated by using partial order planning or graph planning or SAT planning methods.

5.4 Temporal Planning

The plan generation component within the BDI system plays a vital role. In general, planning problems within the BDI system determines the sequence of actions to be performed to achieve the given goal. When it comes to *temporal planning*, time parameter will also come into picture. That means, achieving the given goal within a specific time is crucial. One other technical hurdle that the temporal state space planners need to overcome is that each action could start at any point wherein an infinite number of time points are given. Most temporal planners try to overcome this hurdle by restricting the starting times of action to a small set of decision epochs. *Concurrency* is another important characteristic of temporal planning. The temporal planner should be able to provide a mechanism to cope up with the concurrent actions that can occur at a given *point* of time or *duration* of time. Based on the kind of support provided to handle concurrency, the temporal planning languages are categorized into *temporally simple* and *temporally expressive* languages. If a temporal language L can be used to encode the given problem with the required concurrency, then we can consider it as a temporally expressive language; otherwise L is considered as temporally simple. Temporally simple languages are essentially equivalent to STRIPS in expressiveness. Large class of temporal planners (eg: SAPA, SG plan) are complete only for the temporally simple languages. A complete state space temporal planning algorithm *TEMPO*, takes the advantage of heuristics to achieve high performance.

When we look at temporal planning from an authentication perspective, it is very clear that authentication in communication system is temporal. It means that the authentication is valid only for a given session or for a specified time period. Once the session or specified time period expires, the authentication becomes invalid. Now it is the duty of the communicating entities to establish the authentication again, to continue further. Therefore, from an authentication point of view the temporal planning refers to the time period over which the authentication is valid. In this chapter, we use temporal planning for preparation of a plan so as to reach the destination within a given duration of time.

5.5 Generation of Temporal Plan within BDI

The generation of a plan is an important component in BDI architecture and previous works show how `Propositional` [61, 60], `HTN` (Hierarchical Task Network) [77] as well as `Hybrid` (Classical+ HTN) [34] style planning can be integrated into BDI. BDI system offers flexibility to change the plan or to replan according to changes in the environment to achieve the goal. But it is upto the environment and other constraints to decide whether to generate a new plan or to continue with the existing plan. Temporal planning comes into the picture when goal achievement alone is not sufficient but achieving it within a specified time is also taken care of. Incorporating temporal planning within a BDI framework is a new scenario which needs to be explored. To incorporate temporal planning within a BDI framework firstly the basic BDI architecture needs to be extended to include a temporal action-base wherein each action is annotated with the time it takes to complete that action. The intuition is that in order to execute a plan by an agent in a given environment there should be a corresponding sequence of set of *basic actions* that the agent could perform. It goes without saying that the definition of an agent in [21] as well as the agent model in [87] indirectly refers to having such a set of basic actions. Secondly, we extend the existing classical planning cycle in BDI by incorporating the new temporal action-base so as to achieve temporal planning. Hence, our proposed architecture consists of a `Belief-Base`, `Goal-Base`, `Intention-Base`, `Plan-Library` and `Action-Base` as shown in Figure 5.2.

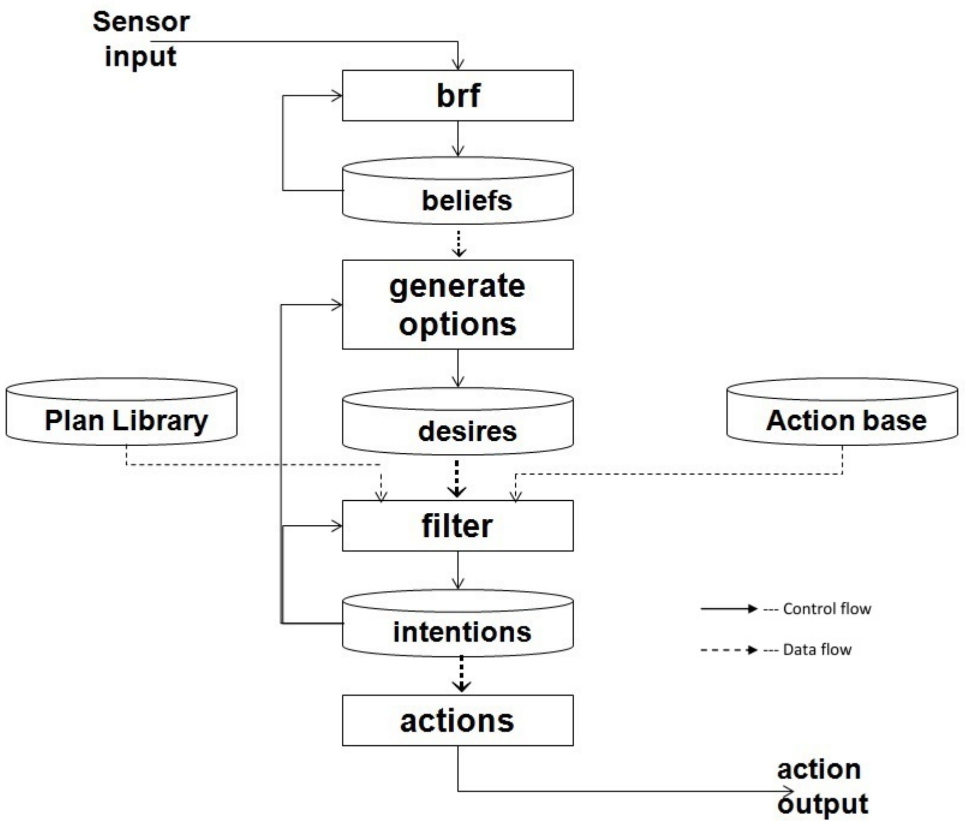


Figure 5.2: Modified BDI Architecture.

In order to generate temporal plan, we have developed four algorithms. In Temporal Planning Algorithm 5.1, first we tried to find the plans for a given goal by making use of plan-library and these plans are called relevant-plans. If relevant-plans are found, then the relevant-plans are processed with belief-base to generate applicable-plans. These applicable-plans are processed with action-base to get Extended-applicable-plans. The time taken is computed for each extended applicable-plan to achieve the given goal. Finally we select Extended-applicable-plans as final plans whose plan-time is less than the given goal-time. If relevant-plans are not found or the Extended-applicable-plans with plan-time less than the goal-time are not found, then we call Generate Plan(1) Algorithm 5.2 to generate the plan.

Algorithm 5.1 Temporal Planning

```
Require: B = belief-base (Current State of Environment)
    PL = plan-library (Predefined Plans)
    G = Goal State including time
    AD = Actions with Duration
Ensure: Plan found or no solution
 1: relevantPlans ← unify(PL, G)
 2: if relevantPlans is empty list then
 3:     plan ← GeneratePlan(PL, B, AD, G)
 4:     if planTime $\leq$ goalTime then
 5:         intention(plan)
 6:     else
 7:         plan not found
 8:     end if
 9: else
10:     applicablePlans ← unify(relevantPlans,B)
11:     extendedApplicablePlans ← substituteGroundTerms
        (applicablePlans,AD)
12:     plans ← calculate(extendedApplicablePlans)
13:     Sort the list based on the time of a plan and remove plans which have time greater than the goal-time
14:     if list is empty then
15:         plan ← GeneratePlan(PL, B, AD, G)
16:         if planTime $\leq$ goalTime then
17:             intention(plan)
18:         else
19:             plan not found
20:         end if
21:     else
22:         intention(list)
23:     end if
24: end if
```

In Generate Plan(1) Algorithm (Algorithm 5.2) we generate target-list (set of initial states) by matching the given goal with belief-base. After generating target-list, we call Generate Plan(2) Algorithm (Algorithm 5.3) for each target(initial state) to generate the plan. Finally it returns the plan-list to Algorithm 5.1.

Algorithm 5.2 Generate Plan(1)

```
Require: belief-base, action-base, plan-library and goal
1: targetList ← getTarget(goal)
2: for all i in targetList do
3:     taget ← targetList[i]
4:     plan ← emptylist
5:     plan ← GeneratePlan(2)(goal)
6:     finalPlanList.append(plan)
7: end for
8: return finalPlanList
```

In Generate Plan(2) Algorithm (Algorithm 5.3), first we generate action-list by matching goal with the action-base, then action-list is processed with plan-library to generate plan-list. For each plan a sub-goal is found by processing with plan-library. It calls Minimum Cost Path Algorithm 5.4 to find the sub plan, with minimum cost from sub-goal to target/initial state. Minimum Cost Path Algorithm 5.4 is called recursively to find the sub plan cost and gets terminated based on the following 3 conditions: (1) If the goal is equal to the initial state, return 0 (2) If the goal is empty list, then there is no path to initial state, return false and (3) If another sub-goal is found. The cost of the applicable action is obtained by matching with the action-base. If more than one plan is found, then return the plan with minimum cost. Else, return empty list. In Minimum Cost

Algorithm 5.3 Generate Plan(2)

```
Require: goal
Ensure: plan or empty list
 1: actionList ← selectActions(goal)
 2: for all i in actionList do
 3:     planList.append(unifyPossibleActions(actionList[i]))
 4: end for
 5: for all i in planList do
 6:     action ← getAction(planList[i],goal)
 7:     if action ≠ emptyList then
 8:         applicable.ActionList.append(action)
 9:     end if
10: end for
11: for all i in planList do
12:     temp ← emptyList
13:     t ← minCostPath(getNewGoal(planList [i], goal),temp1)
14:     if t = 0 then
15:         t ← True
16:     else if t = False then
17:         continue
18:     else
19:         minValue.append(float(t)+ gettime(applicableActionList[i]))
20:         temp1.append(applicableActionList[i])
21:         temp.append(temp1)
22:     end if
23: end for
24: if len(minValue) > 1 then
25:     min ← float(minimumCost(minVlaue,index))
26:     finalPath.append(temp[0])
27: else if len(minValue) = 1 then
28:     min ← float(minValue[0])
29:     finalPath.append(temp[0])
30: else
31:     finalPath.append(emptyList)
32: end if
```

Path Algorithm (Algorithm 5.4), first we check whether the given goal and initial state are same; if both are same, then it returns 0, meaning that no plan and no time required to reach goal from initial state. If goal is empty, then it returns empty list, it means, no plan is required. If goal and initial state are different and goal list is not empty, then plan-list is generated by processing given action list with plan-library. After that it tries to find the sub-goal for each plan in the plan-list and it recursively itself to find the sub plan, from sub-goal to target/initial state with minimum cost and it gets terminated based on the following 3 conditions.

1. If the goal is equal to the initial state, return 0.
2. If the goal is empty list, then there is no path to initial state, return false.
3. If another sub-goal is found.

The cost of the applicable action is obtained by matching with the action-base. If more than one plan is found, then return the plan with minimum cost. Else, return empty list.

Algorithm 5.4 Minimum Cost Path

```
Require: goal and path
Ensure: value or False
 1: if $target = goal$ then
 2:     return 0
 3: else if $g = emptyList$ then
 4:     return []
 5: else
 6:     $actionList \leftarrow$ selectActions($goal$)
 7:     for all $i$ in $actionList$ do
 8:         $planList$.append(unifyPossibleActions($actionList$[i]))
 9:     end for
10:     for all $i$ in $planList$ do
11:         $action \leftarrow$ getAction($planList$ [$i$], $goal$)
12:         if $action \neq emptyList$ then
13:             $applicable.ActionList.$append$(action)$
14:         end if
15:     end for
16:     for all $i$ in $planList$ do
17:         $temp \leftarrow emptyList$
18:         $t \leftarrow$ minCostPath(getNewGoal($planList$ [$i$], $goal$),$temp1$)
19:         if $t = 0$ then
20:             $t \leftarrow True$
21:         else if ($t = emptyList$) or ($t = False$) then
22:             return $False$
23:         else
24:             minValue.append(float(t) + gettime(applicableActionList[i]))
25:             $temp1$.append($applicableActionList$[i])
26:             $temp$.append($temp1$)
27:         end if
28:     end for
29:     if $len$(minValue) $> 1$ then
30:         $min \leftarrow$ float(minimumCost($minVlaue,index$))
31:         $path$.append($temp$[0])
32:     else if $len(minValue) = 1$ then
33:         $min \leftarrow$ float($minValue$[0])
34:         $path$.append($temp$[0])
35:     else
36:         return $False$
37:     end if
38: end if
```

The flow diagram of Temporal planning within BDI system, which is a combination of all the four algorithms is as shown in the Figure 5.3

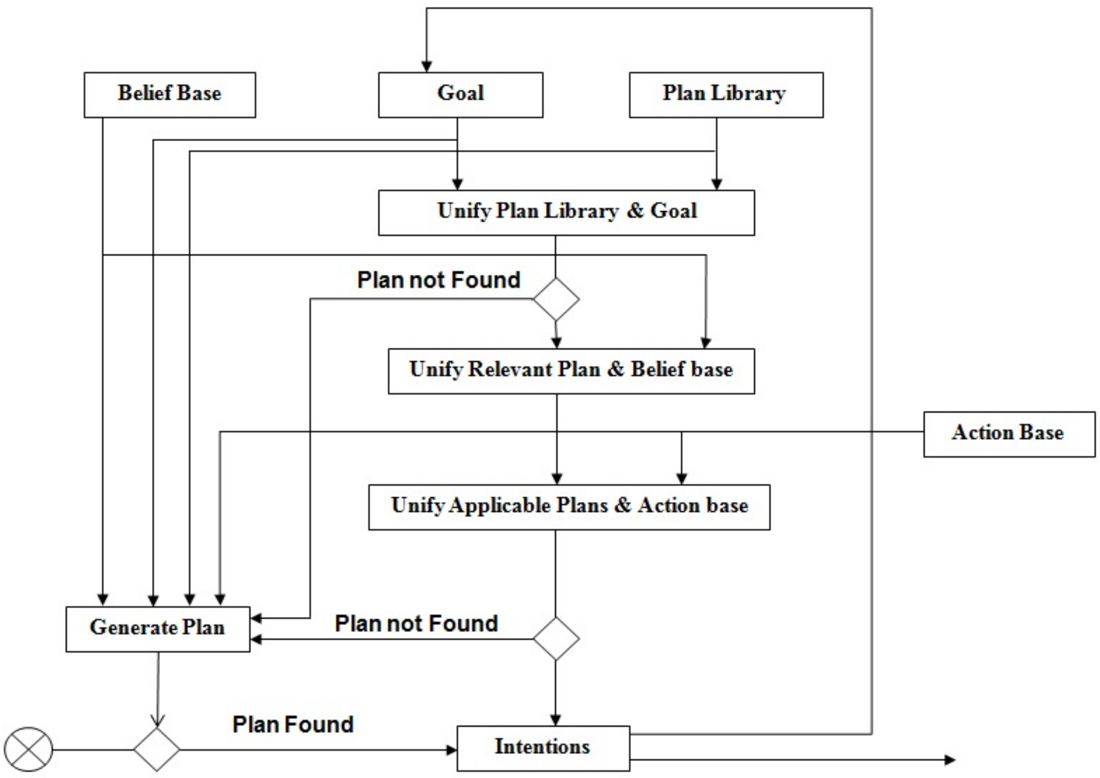


Figure 5.3: A Flow Diagram for Modified BDI Architecture.

5.6 Temporal Planning with BDI - A case study

In order to study and evaluate Temporal Planning within BDI, we consider the following problem which is a modified version of the problem given in [37]. *A group of students need to go from Gachibowli to Bangalore (two cities in India). There is a car rental company, which has two cars i.e., car1 and car2. Car1 can go to the railway station or the airport and car2 can go to Bangalore or to the airport. So there is a possibility of taking a train from the railway station to Bangalore or a flight from the airport to reach Bangalore. These options can be categorized as follows*: *(1) rent a car and drive from Gachibowli to Bangalore directly (2) rent a car from Gachibowli to Shamshabad (name of the airport) and then go by flight from Shamshabad to Bangalore. Each vehicle takes some time to reach particular place. User wants to reach the destination by imposing time constraints i.e., the total time to reach the destination be not more than threshold time(such as to be* *in Bangalore within two hours).*

The travel problem considered above contains a specified goal (reaching to Bangalore) and a constraint (within two hours or 120 minutes). In order to find a solution that is good with respect to time, we need to track time of (sub) goals. The planner has to generate a plan that is optimal and satisfies the user constraints. In real world problems, planning with time is very important wherein performing poor action within time is preferable than performing planned action which is too late. Normally solving a problem by human beings under extreme pressure downsize the quality of solution and also solving the problem on time. Whereas in classical planning, the quality of a plan is based on the length of the plan in temporal planning, the user is more interested in improving the temporal quality of a plan.

1. at(gachibowli)
2. at(car1, gachibowli)
3. at(car2, gachibowli)

Table 5.2: Belief-base

1. Drive(car1, gachibowli, secunderabad)	Time = 210
2. Drive(car1, gachibowli, shamshabad)	Time = 60
3. Drive(car2, gachibowli, shamshabad)	Time = 90
4. Drive(car2, gachibowli, bangalore)	Time = 420
5. Fly(shamshabad, bangalore)	Time = 90
6. Train(secunderabad, bangalore)	Time = 150

Table 5.3: Action-base

We can formulate the facts or knowledge of the problem given above in the form of a belief-base as shown in Table 5.2 and is usually referred to as belief literals. In our current scenario we can also have as belief literals **not** at(car1, Bangalore), **not** at(car2, Bangalore), etc. From the problem statement we can also formulate the set of actions that can be applied in the environment in the form of Action-Base as shown in Table 5.3. It can be seen from the action-base that each action is attached with a temporal duration. For instance, Drive(Car1, Gachibowli, Secunderabad) Time=210 means that the action Drive takes 3.5 hours to reach Secunderabad from Gachibowli. *Goal* is a state of the system which the agent wants to bring about which for our problem is `at(Bangalore)`. In a similar manner one can interpret the other actions. *Plan-Library* contains the predefined plans as shown in Table 5.4. The plans in the plan-Library consists of a head and a body. The head of a plan consists of a triggering event and a context. The triggering event denotes the purpose of the plan. The context in the head of a plan is a conjunction of belief literals.

The context needs to be satisfied if the plan is to be executed and it should also be the case that the context is a logical consequence of the agent's current set of beliefs.

1. (Drive(X, Y, Z)	:	at(Y), at(X, Y) ← + at(Z), - at(Y), - at (X, Y), + at(X, Z))
2. (Train(X, Y)	:	at(X) ← + at(Y), - at(X))
3. (Fly(X, Y)	:	at(X) ← + at(Y), - at(X))

Table 5.4: Plan-library

If we consider the plans in Table 5.4, then the expression on the left side of the arrow is considered as the *head* of the plan and that on the right side is considered as the *body* of the plan. The expression to the right side of the colon in the head of a plan is considered as the *context.* The plan body may consist of +, -, !, ? which represents addition of the belief or goal, deletion of belief or goal, achievement goal and test goal respectively. For instance when the following plan (Plan 1 in Table 5.4) is executed.

$$\begin{gathered}(Drive(X, Y, Z) : at(Y), at(X, Y) \longleftarrow \\ +at(Z), -at(Y), -at(X, Y), +at(X, Z))\end{gathered} \tag{5.1}$$

it should result in the agent believing the following (1) he/she is at particular city Z (2) he/she is no longer in the city Y (3) the car is no longer at Y and (4) the car is at Z. For instance in our case the BNF [35] of a Trigger is given as

Trigger ::= Belief | *Goal* | *Action* which includes an action apart from belief and goal as in the original version. Unless stated explicitly the triggering events always have a (+) attached to them.

5.6.1 Solution

The solution to the given problem is determined by making use of the four algorithms explained in the previous section and `Belief-base`, `Action-base`, `Plan-library`. In Algorithm 5.1, first the given goal '`at(bangalore)`', is matched with the heads (context part) of plans in the Plan-library. This is done by matching predicate name (in this case `at`) and *number* of parameters of goal (one in this case) irrespective of the type of parameters i.e., whether the parameter is a ground term or a variable. The plans which are found are called *relevant-plans* (Line 1 of Algorithm 5.1). If the relevant-plan-list is an empty list, then we try to find a plan by using GeneratePlan[2]. GeneratePlan returns a plan with an optimal time which may be less than the user specified time. The plan time is checked with the user time (goal-time) and if its time is less than the goal-time, then we put that plan in the intention stack for execution. Otherwise, return null i.e., plan not found (Line 2-8 of Algorithm 5.1).

Once the relevant-plans are found, check whether these plans satisfy the context (preconditions) or not, because relevant-plans may have variables in the precondition. This is done by unifying relevant-plans with the belief-base through the substitution of belief literals in place of variables (Line 10 of Algorithm 1). To illustrate this procedure with an example, consider that the belief-base contains the following beliefs 1. at (gachibowli), 2. at (car1, gachibowli), 3. at (car2, gachibowli) and 4. at (car3, gachibowli) and suppose we have a single relevant-plan like

$$\textit{at(Bangalore): at(X), at(Z,X)} \longleftarrow \textit{Drive(Z,X,P),Fly(P,Bangalore)}$$

which can be interpreted as *If someone drives Z from X to P and flies from P to*

[2]We explain the working of the GeneratePlan later.

bangalore then he/she is at bangalore provided he/she was at X and Z was at X. For the example given above, the variable X takes X/gachibowli and Z takes Z/car1, car2, car3. After substitution we get the following plans.

1. *at(Bangalore): at(gachibowli), at(car1, gachibowli)* $\longleftarrow$ *Drive (car1, gachibowli, P), Fly(P, Bangalore)*

2. *at(Bangalore): at(gachibowli), at(car2, gachibowli)* $\longleftarrow$ *Drive (car2, gachibowli, P), Fly(P, Bangalore)*

3. *at(Bangalore): at(gachibowli), at(car3, gachibowli)* $\longleftarrow$ *Drive (car3, gachibowli, P), Fly(P, Bangalore)*

The plans given above satisfy the `context` (because at(gachibowli), at(car1,gachibowli)) and at(car2,gachibowli) are all logical consequences of the belief-base) and therefore these plans are called `applicable-plans`. In the applicable plans, there may be variables in the plan body. Substitute these variables with the possible ground terms by unifying `action` and `applicable-plans`(Line 11 of Algorithm 5.1). To make things clear, suppose that the action is given as in Table 5.3. Consider the above set of actions and plans as given before we can see that plans have variable `'P'` in their plan body. Hence possible values are `P`/shamshabad. After substituting values for `P`. in applicable-plans we get the following plans.

1. *at(Bangalore): at(gachibowli), at(car1, gachibowli)* $\longleftarrow$ *Drive (car1, gachibowli, shamshabad), Fly(shamshabad, Bangalore)*

2. *at(Bangalore): at(gachibowli), at(car2, gachibowli)* $\longleftarrow$ *Drive (car2, gachibowli, shamshabad), Fly(shamshabad, Bangalore)*

3. *at(Bangalore): at(gachibowli), at(car3, gachibowli)* $\longleftarrow$ *Drive (car3, gachibowli, shamshabad), Fly(shamshabad, Bangalore)*

These plans are called `Extended-applicable-plans`. Extended-applicable-plans may include plans that are not necessary. So, remove the unwanted plans by checking for consistency between the actions in the plan and the action-base. For instance, in the third plan given above, the action `Drive(car3, gachibowli, shamshabad), Fly(shamshabad, Bangalore)` is unwanted because such an action is absent in the action-base of Table 5.3. and therefore we end up with the following plans in our plan-base;

1. *at(Bangalore): at(gachibowli), at(car1, gachibowli)* $\longleftarrow$ *Drive (car1, gachibowli, shamshabad), Fly(shamshabad, Bangalore)*

2. *at(Bangalore): at(gachibowli), at(car2, gachibowli)* $\longleftarrow$ *Drive (car2, gachibowli, shamshabad), Fly(shamshabad, Bangalore)*

After removing the unwanted plans, calculate the time for all the remaining plans. This can be done by selecting an item in the plan body which may either be a sub-goal or an action. If the item is a sub-goal, then we need to find plans and process the plan body until the selected item is an action. If item is an action, then search in the action-base for time. Calculate the time for all the plans and sort them. The plans that have less than or equal time to the goal-time are put in the intention for execution. The time taken for plan1 and plan2 are 3 hours 2 and a half hours respectively. If the user goal-time is > 4 hours, then plan1 and plan2 are selected. If goal-time is < 3 hours, then only one plan i.e., plan2 is selected. If user specified time is 2.5 hours, then no plan is selected an in that case it will find a plan using `GeneratePlan`, that is Algorithm 2.

Before applying Algorithms 2,3, and 4 to the problem to find solution, let us first understand the functions used in those algorithms. Algorithm 5.2 takes all the input and set the initial state (target-list) using the getTarget function (Line1 of algorithm 2). This is done by matching the goal with the belief-base. For this to happen the goal predicate as well as the number of its parameters should be the same as that of a belief in the belief-base. For example, if the goal is to be at(bangalore), then targetList contains at(gachibowli), the first belief in the belief-base of Table 5.2. (the predicate at of at(bangalore) matches with that of at(gachibowli) and the number of parameters is also the same which in this case is one). If belief-base contains another item like at(ranchi), then targetList will have at(gachibowli), at(ranchi). Algorithm 2 provides a high level description of our BDI plan generation method. Internally it uses Algorithm 3 to generate all paths (each path is a plan) from goal to the target/s in the targetList (Line 5 of

Algorithm 2) and this is done at two different levels (a) when the list of relevant-plans is empty (Line 3 of algorithm 1) or (b) when there is no plan in the list of extendedApplicablePlans that matches with the user specified time (Line 15 of algorithm 1).

The main idea behind Algorithm 3 is that it takes a goal and returns an optimal plan/path if it finds one, otherwise it returns, saying 'plan cannot be obtained'. We start by explaining the different functions used in the algorithm. The selectActions function (Line 1 of Algorithm 3) takes goal as its parameter and searches in the action-base (Table 5.3.) for actions that have the same parameter name as that of the goal parameter. For example, if goal is at(bangalore), then the corresponding actions with bangalore as a parameter are returned which, in this case will be (1) Drive(car2, gachibowli, bangalore) (2) Fly(shamshabad, bangalore) and (3) Train(secunderabad, bangalore). UnifyPossibleActions function (Line 3 of Algorithm 3) takes action as parameter. This function first tries to match an action from the actionList (Line 1-2 of Algorithm 3) with a plan in the plan-library of Table 5.4. This is done by matching action name and number of parameters with the head of the plan. Secondly, it unifies the plan and action by instantiating the variables appearing in the plan. For example, if action = Fly (shamshabad, bangalore) then unifyPossibleActions returns the following plan

$$\begin{aligned} Fly(shamshabad, bangalore) : at(shamshabad) \longleftarrow \\ +at(bangalore), at(shamshabad) \end{aligned} \tag{5.2}$$

after unification with the 3rd plan (y(X, Y) : at(X) ⟵ +at(Y), -at(X)) of Table 5.4.

The getAction function (Line 6 of Algorithm 3) takes goal and plan as parameters and checks whether this plan has goal in the plan body as a positive literal. If goal is present in the plan body, then it returns the action/s that led to this goal, otherwise, returns empty list (Lines 6-8 of Algorithm 3). For example, in the plan given in (2), at(bangalore) appears as a positive literal and therefore the action Fly(shamshabad, bangalore) is returned. getNewGoal is a function to generate new-goal. It takes a goal and a plan as inputs and the goal is matched with the positive literals in the plan body. If there is no match, that means, there is no new-goal. Otherwise, select a literal from the context of the plan having same-goal name and same number of parameters and also that does not hold after execution. This becomes the new-goal. For example, if goal is at(bangalore) and plan is Fly(shamshabad, bangalore): at(shamshabad) $\longleftarrow$ +at(bangalore), -at(shamshabad), then new-goal is at(shamshabad).

minCostPath (Algorithm 4) is a recursive function that returns the (sub) plan path cost if it finds one, otherwise, returns false. It takes a goal and a variable (called, path) as its inputs. The variable is used for storing the plan path. Termination condition of this function depends on (a) whether the goal is equal to the initial state in which case the function returns 0 (b) whether the goal is empty list in which case there is no path to reach the initial state and hence the function returns false (c) whether there are other states from which the initial state is reachable. The function is applied recursively until the target or a dead state (no states which achieve this state) is reached. The getTime takes an action as parameter from a list of applicableActionList and the action name is checked in the action-base. If there is a match, then the corresponding time for that action is returned otherwise return null (Lines 19-21 of Algorithm 3). For example, if Fly(shamshabad, bangalore) is a parameter for getTime, then the action name Fly (shamshabad, bangalore) is searched in the action-base of Table 5.3 and 90 is returned. minimumCost function returns the minimum cost value of the (sub)plan and index value in the list by taking list of plan cost and variable to store index value.

Now we explain how plan is generated when there are no relevant-plans generated by Algorithm 1. When a goal (at(bangalore) 3) is given, initially it is separated into two parts, (a) *goal* which is (at(bangalore)) and (b) *goal-time* which is (3). The goal is matched with plan-library Table 5.4. Since there are no matching (relevant) plans, we try to build a plan using Algorithms 5.2,5.3 and 5.4. The question of matching the goal initially with the belief-base (Table 5.2) does not

arise because the goal in our case is the selected-event in the traditional BDI architecture[2]. In our work the starting point is the selected event which becomes the initial goal.
Since there are no matching plans, we have empty list of relevant-plans. In order to generate a plan we first make use of the *getTarget* function of Algorithm 5.2, so as to find the initial state (target). This is needed because given a goal state we are trying to build paths/plans that reach this goal state. We cannot go on building a plan from a goal state unless and until we know the initial state which tells us that we have to stop building the plan or that we have reached the target/initial state.[3] Therefore, to find the initial state getTarget function searches in the belief-base for a belief which have same-goal name and same number of parameters as that of the goal. So in our problem, *target = at(gachibowli)* is set. Since we have only a single item in the target-list and an emptylist is assigned to plan (Lines 2-4 of Algorithm 5.2), we go on to Line 5 of Algorithm 5.2 which internally calls Algorithm 5.3 and which in turn gets executed by taking goal (at(bangalore)) as parameter. If there are multiple targets, then we will call Algorithm 3 for each such target.

The selectActions function as defined in the previous section (Line 1 of Algorithm 5.3) selects the following three actions `Drive(car2, gachibowli, bangalore)`, `Fly(shamshabad, bangalore) and Train(secunderabad, bangalore)`. These actions are unified with the plans in the plan-library using the unifyPossibleActions function (Lines 2-3 of Algorithm 5.3) and the output is appended in planList. The planList consists of the following plans.

$$\text{(P1) } \textit{Drive(car2, gachibowli, bangalore) : at(gachibowli),}$$
$$\textit{at(car2, gachibowli)} \longleftarrow \textit{+at(bangalore), at(gachibowli),}$$
$$\textit{at(car2, gachibowli), +at(car2, bangalore)}$$
$$\text{(P2) } \textit{Fly(shamshabad, bangalore) : at(shamshabad)} \longleftarrow$$
$$\textit{+ at(bangalore), at(shamshabad)}$$
$$\text{(P3) } \textit{Train(secunderabad, bangalore) : at(secunderabad)} \longleftarrow$$

[3]This approach is similar to the way the graph plan is searched backward for a plan[1].

+ at(bangalore), at(secunderabad)

It should be noted that we are selecting actions without thinking whether it will reach the goal or not. For example, if the action Drive(car2, bangalore, gachibowli) is present in the action-base then it will also be picked as we are only checking action name and number of parameters. Now by performing this action, we will never be able to reach bangalore. For this reason, actions are unified with the corresponding plans in the plan-library. After `unification` we check all the plans in the planList to see whether the goal (at(bangalore)) is present in the plan body as a positive literal (The *getAction* function is used here). If a plan satisfies this constraint, then append the action which led to this goal to applicableActionList, otherwise discard the plan (Lines 5-10 of Algorithm 5.3).

In our example, all plans (P1, P2 and P3) in the planList have (at(bangalore)) as a positive literal and therefore all the actions that lead to the goal (at(bangalore)) are added to applicableActionlist. In this case the applicableActionlist consists of (a) Drive(car2, gachibowli, bangalore) (b) fly(shamshabad, bangalore) and (c)Train(secunderabad, bangalore). At this time, we are not selecting any one action which has less time but process all applicable-plans because there might be a possibility of getting an optimal plan by selecting an action which takes more time initially. The next step (Lines 11-13 of Algorithm 5.3) is to get inputs for the getNewgoal function. getNewgoal function takes each Plan in the planList, the current-goal and a variable (temp1) as inputs. It then returns a new-goal as a value to the minCost algorithm.

Let us see how this is done. Suppose that the getNewgoal function selects the rst plan (P1) in the applicableActionlist, the current-goal and a variable called temp1 which is given as follows,

getNewGoal(P1, at(bangalore), temp1)

Then we get at(gachibowli) as the new-goal because by the definition of the get-Newgoal function given in the previous section it takes a goal and a plan as inputs and the goal is matched with the positive literals in the plan body. Here we have at(bangalore) as the goal and at(bangalore) appears as a positive literal in the body of `P1`. So there is a match. Then we should select a literal from the *context* of `P1` having same-goal name (which here is `at`) and same number of parameters (which here is one) and also that does not hold after execution (which here is -at(gachibowli). So finally, we get the values of minCostPath as

$$t = minCostPath(at(gachibowli), temp1)$$

Now apply Algorithm 5.4. This algorithm takes two parameters (1) new-goal which here is (at(gachibowli)) and (2) a new variable to store path. In this case, it returns 0 because the new-goal which is at(gachibowli) is equal to the target (Lines 1-2 of Algorithm 5.4)[4]. Next, the time taken to reach bangalore using `P1` is calculated as the sum of the values returned by minCost and the function `gettime` (Line 19 of Algorithm 5.3). Therefore, the total time of the plan is 0 + `getTime`(Drive(car2, gachibowli, bangalore)) = 0 + 420 = 420 and this value is appended to minValue. The path (Drive(car2, gachibowli, bangalore) and time (420) is stored in *temp1* and appended to the empty list *temp* (Lines 20-21 of Algorithm 5.3). There are two more plans in the planList which we need to process so as to get the best possible plan to reach the goal at(bangalore). `Plan2 (P2)` is selected and we repeat the same procedure as above to get values for getNewGoal and minCostPath.

$$getNewGoal(P2, at(bangalore), temp1)$$

We get at(shamshabad) as the new-goal because it has the same name (at) as well as the same number of parameters (one) as the current-goal (which is at(bangalore) and does not hold after execution of `P2` (i.e., it appears as -at(shamshabad) in P2). From this, we get values for minCostPath as

$$t = minCostPath(at(shamshabad), temp1)$$

[4]Remember that we set target as at(gachibowli) initially.

minCostPath checks whether the new-goal (at(shamshabad)) is equal to target or empty list. It is neither empty nor equal to target and therefore Algorithm 5.4 will select all the actions related to this new-goal (Lines 1-6 of Algorithm 5.4). There are two possible actions to reach the new-goal (1) `Drive` (car1, gachibowli, shamshabad) and (2) `Drive`(car2, gachibowli, shamshabad). Select the actions and apply the unification procedure as explained earlier (Lines 7-15 of Algorithm 5.4)[5]. We get the following plans to be appended in the planList.

$$\begin{gathered}
(\mathrm{P1}')\ \textit{Drive(car1, gachibowli, shamshabad) : at(gachibowli),}\\
\textit{at(car1, gachibowli)} \longleftarrow \textit{+at(shamshabad),}\ \ \textit{at(gachibowli),}\\
\textit{at(car1, gachibowli), +at(car1, shamshabad)}\\
(\mathrm{P1}'')\ \textit{Drive(car2, gachibowli, shamshabad) : at(gachibowli),}\\
\textit{at(car2, gachibowli)} \longleftarrow \textit{+at(shamshabad),}\ \ \textit{at(gachibowli),}\\
\textit{at(car2, gachibowli), +at(car2, shamshabad)}
\end{gathered}$$

Both plans P1′ and P1″ have at(shamshabad) (which is the current-goal) as positive literals in the plan body and therefore all actions that lead to at(shamshabad)) are added to applicableActionlist. In this case, the actions are (a)`Drive`(car1, gachibowli, shamshabad) and (b) `Drive`(car2, gachibowli, shamshabad). Next, we check for the parameters of the getNewGoal function for both the plans and this is given as

$$\begin{gathered}
\textit{getNewGoal}(P1',\ \textit{at(shamshabad), temp1)}\\
\textit{getNewGoal}(P1'',\ \textit{at(shamshabad), temp1)}
\end{gathered}$$

We get at(gachibowli) as the new-goal because it has the same name (at) as well as the same number of parameters (one) as the current-goal (which is at(shamshabad)) and which does not hold after execution of P1′ and P1″ (see definition of getNewGoal). Hence the value for minCost for both the plans is

$$t = \textit{minCostPath(at(gachibowli), temp1)}$$

[5]It should be noted that while processing P2 we need to work within Algorithm 5.4 so as to derive further possible actions, applicable-plans etc. whereas for P1 most processing was done within Algorithm 5.3. This explains the almost common structure for both algorithms

The output value of mincostPath (Algorithm 5.4) is 0 because the new-goal at(gachibowli) is equal to the target (Lines 19-23 of Algorithm 5.4) in both cases. Now, the time taken to reach shamshabad using $P1'$ is calculated as the sum of the values returned by minCost cost and actionTime(Drive(car1, gachibowli, shamshabad)) which is 0 + 60 =60 (Line 24 of Algorithm 5.4). This value is added to the minValue list and path added to temp (Lines 24-26 of Algorithm 5.4). Similarly the output value of mincostPath for $P1''$ is 0 and the time taken to reach shamshabad using this plan is calculated as sum of the values returned by minCost cost and actionTime(Drive(car2, gachibowli, shamshabad)) which is $0 + 90 = 90$. In the next step, Algorithm 5.4 checks whether the length of minValue is greater than 1 or equal to 1 or is an empty list. In this case, minValue has two values (60 and 90) and therefore the length is greater than one. Hence, min is assigned the smaller value (i.e., 60) and the corresponding path (in this case Drive(car1, gachibowli, shamshabad)) is appended to the path variable path (Lines 29-31 of Algorithm 5.4). This value and path is returned to Algorithm 5.3 to calculate the total time taken to reach bangalore. Remember that we have only calculated the best possible time to reach shamshabad from gachibowli (which is 60) and the corresponding action (Drive(car1, gachibowli, shamshabad)) that achieves this. We still have to calculate the time taken from shamshabad to bangalore and the corresponding path in order to calculate the time from gachibowli to bangalore.

Time taken to reach bangalore from gachibowli is sum of the minCost value returned by Algorithm 5.4 (which is 60) and gettime (applicableActionList[i]) which is (y(shamshabad, bangalore)) and for which the value is 90. So the total time taken is $60 + 90 = 150$. This value is added to minValue and path is appended to the temp list (Lines 19-21 of Algorithm 5.3). Here, one should note that in our applicableActionlist which was created initially from the original set of plans (P1, P2 and P3), we had only three actions namely (a) Drive(car2, gachibowli, bangalore) (b) y(shamshabad, bangalore) and (c) Train(secunderabad, bangalore). Of these three, only the second action is applicable in the current scenario. In the next step, Algorithm 5.3 checks whether the length of minValue is greater than 1 or equal to 1 or is an empty list. In this case, minValue has two values (420 and 150) because 420 is the value min has stored from previous processing of P1. Therefore, the length is greater than one and min is assigned the smaller value (i.e., 150) and the corresponding path (in this case Drive(car1, gachibowli, shamshabad)) is appended to the path variable path (Lines 24-26 of Algorithm 5.3).

Now we select the third and final plan `P3` which is

$$\begin{gathered}\textit{Train(secunderabad, bangalore) : at(secunderabad)} \longleftarrow \\ +\ \textit{at(bangalore), at(secunderabad)}\end{gathered}$$

We repeat the same procedure as above to get values for getNewGoal and minCostPath as follows

$$\textit{getNewGoal(P3, at(bangalore), temp1)}$$

We get at(secunderabad) as the new-goal and the values for minCostPath as

$$t = \textit{minCostPath(at(secunderabad), temp1)}$$

Apply Algorithm 5.4, taking secunderabd as the new-goal and a variable as parameters. As mentioned, minCost algorithm is a recursive algorithm and it terminates only if the goal is equal to target or null. Otherwise it selects actions to reach the new-goal and process all actions until it reaches target or null. In this case, at(secunderabad) is not equal to target and therefore algorithm 4 will select all the actions related to this new-goal. There is one action which can reach new-goal and that is Drive(car1, gachibowli, secunderabad). After unification (Lines 7-15 of Algorithm 5.4) get the following plan

$$\begin{gathered}(\mathrm{P1}''')\ \textit{Drive(car1, gachibowli, secunderabd) : at(gachibowli),} \\ \textit{at(car1, gachibowli)} \longleftarrow \textit{+at(secunderabd), at(gachibowli),} \\ \textit{at(car1, gachibowli), +at(car1, secunderabd)}\end{gathered}$$

to be appended in the planList. The plan P1$'''$ have at(secunderabad) (which is the current-goal) as a positive literal in the plan body and therefore all actions that lead to at(secunderabad) are added to applicableActionlist. In this case, there is only one action i.e., Drive(car1, gachibowli, secunderabad). Next, we check for the parameters of the getNewGoal function for the plan and this is given as

$$\textit{getNewGoal(P}1''', \textit{at(secunderabd), temp1)}$$

We get at(gachibowli) as the new-goal because it has the same name (at) as well as the same number of parameters (one) as the current-goal (which is at(secunderabad)) and which does not hold after execution of P1$'''$ (again see definition of getNew-Goal). Hence the value for minCost for both the plans is

$$t = \textit{minCostPath(at(gachibowli), temp1)}$$

Apply mincostPath algorithm taking a variable and at(gachibowli) as new-goal and the returned value will be 0 as the new-goal is equal to the target (Lines 19-23 of Algorithms 5.4). Now the time taken to reach secunderabad from gachibowli using P$'''$ is calculated as the sum of the values returned by minCost and action-Time (Drive(car1, gachibowli, secunderabad)) which is 0 + 210 = 210 (Line 24 of Algorithm 5.4). This value is added to the minValue list and path added to temp (Lines 24-26 of Algorithm 5.4). In the next step, Algorithm 5.4 checks whether the length of minValue is greater than 1 or equal to 1 or is empty list. In this case, minValue has only one value and therefore min is assigned the value 210. This value along with the path is passed back to algorithm 5.3 to calculate the total time taken to reach bangalore. Time taken to reach bangalore from gachibowli is sum of the minCost value and actionTime which is equal to 210 + getTime (Train(secunderabad, bangalore)) = 210 + 150 = 360 (Line 13 of Algorithm 5.3). This value is added to minValue and paths to temp list of Algorithm 5.3.

Now in the next step, Algorithm 3 checks whether the length of minValue is greater than 1 or equal to 1 or is an empty list. In this case, min list has two values

(150 and 360) where 150 is the previous value from processing P2 and therefore the length is greater than one. Hence, min is assigned the smaller value (i.e., 150) and the corresponding paths (in this case Drive(car1, gachibowli, shamshabad), Fly(shamshabad, bangalore)) is appended to nalPath. In this way, we generate a plan and the path/plan along with time is passed to Algorithm 5.1 (either at Line 3 or Line 15). While generating the plan we are not taking into consideration of the time given by the user. We simply generate and return the optimal plan path and then the plan time is checked with the user given goal-time. If the plan time is less than or equal to the goal-time, then the plan is put in the intention stack for execution. If that is not the case, then 'plan not found' is returned. Figure 5.4 is complete solution of the travel problem. The numeric values adjacent to the action name represent the time required to complete that action. The root node is the goal node and leaf node is the initial state. Levels 1, 3 and 5 denote the propositions true at that time/level and levels 2 and 4 denote the actions. In Fig 5.4, ban, sc, sh, and gb mean bangalore, secunderabad, shamshabad and gachibowli respectively.

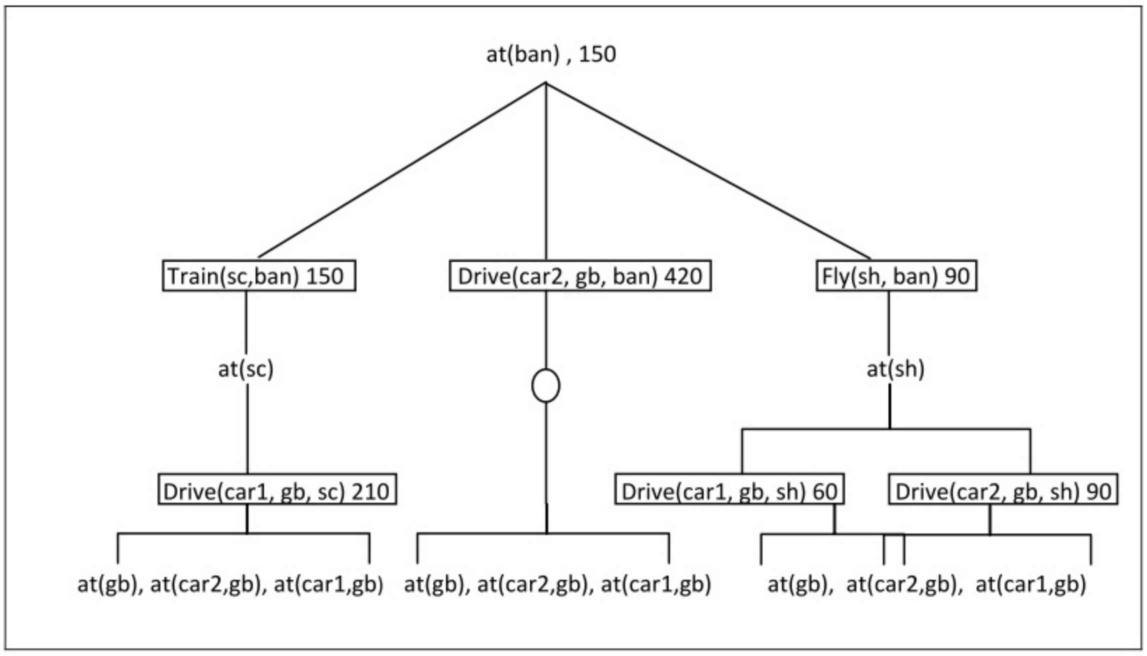


Figure 5.4: Complete solution of the travel problem

5.7 DNSSEC with Temporal BDI

As mentioned earlier the DNSSEC protocol was designed to provide security to the DNS system and to perform this task DNSSEC protocol uses a technique called *chain of trust*. In the chain of trust construction, the DNSSEC protocol actually ensures the identity/credentials of each node that comes across in the chain, with its parent node. Finally, when it reaches to the point of trust (i.e. the root node) the process is stopped.

In this Section , we extend the concept of temporal planning in BDI as discussed in the previous section to that of DNSSEC. We try to analyse the DNSSEC protocol in the background of a BDI model so as to reason about the construction of chain of trust and thereby utilize the authentication results effectively. The DNSSEC protocol tries to build a chain of trust from the domain under consideration to the point of trust, also called, *anchor point* (in our case it is *root* node). The actual task carried during the construction of chain of trust is verification of the credentials of child nodes with its respective parent node. If we observe closely the process of chain of trust construction, it is very clear that it generates additional requests/queries to ensure the authentication and integrity.

Obviously this process will also take additional amount of time to construct chain of trust. The performance of DNSSEC protocol can be improved, if we can save the successful chain of trusts into cache memory of local server for a valid time period. The idea is similar to that of saving of an IP address into cache memory in present DNS system scenario. In our case, we save the chain of trust of nodes in terms of belief-axioms into the belief-base. Next time, before one starts constructing the chain of trust for a particular node, we first check the belief-base to see whether it contains the belief-axiom for a given node. If the belief-base contains the belief-axiom for the given node, then we will not construct a new chain of trust as we directly believe in that node. Otherwise, we have to construct a new chain of trust from the given node to the root node before starting to believe it. If we can effectively utilize the previous results that were obtained while constructing the chain of trust, probably upto some extent we can save the time as well as avoid the generation of additional requests to construct chain of trust, which in turn would reduce the *internet traffic*. Therefore, the effective utilization of results obtained from previous queries would definitely improve the performance of the DNSSEC protocol. The research problem addressed in this chapter is that of effective utilization of results related to chain of trust so as to avoid the number of constructions related to chain of trust.

5.8 Performance Enhancement Method for DNSSEC with Temporal BDI

In order to make use of the BDI model in the analysis of DNSEC-like protocol first we have to understand the protocol and then develop *belief-base*, *action-base* and the *plan-library* corresponding to it. Once we get the belief-base, action-base, and plan-library, then we can apply appropriate algorithms to achieve our goal. The steps followed to analyze DNS and DNSSEC within BDI can be given as follows:

1. The IP address for a given domain name is checked in the belief-base and if it is found then go to step 3.

2. The DNS query resolution procedure is called to find the IP address for a given domain name. If the IP is detected successfully, then continue, otherwise stop the process and exit.

3. If belief-base does not contain the belief-axiom for a given IP address, then call the DNNSSEC procedure to construct the chain of trust before believing the given IP address. Otherwise, simply declare that the given IP address can be believed.

4. If the construction of chain of trust was successful, then add the belief-axiom corresponding to the given IP address and also add the belief-axioms corresponding to all the nodes that come across in the chain of trust into the belief-base.

The fourth step in the above procedure is crucial in our analysis because it improves the the overall performance of the system by avoiding the reconstruction of the chain of trust for the domains to which it has already constructed.

From Figure 5.5 it is very clear that the DNS query resolution is a top-down process, that is, the client node starts sending request to the server *root* node and then comes down one level at a time until it gets the required response. It is also clear from Figure 5.5 that the chain of trust construction of the DNSSEC protocol is a bottom-up process. It means that the client node starts sending request to the parent of the node from which it got the response, then to the parent of the parent node and this process continues until the root node is reached.

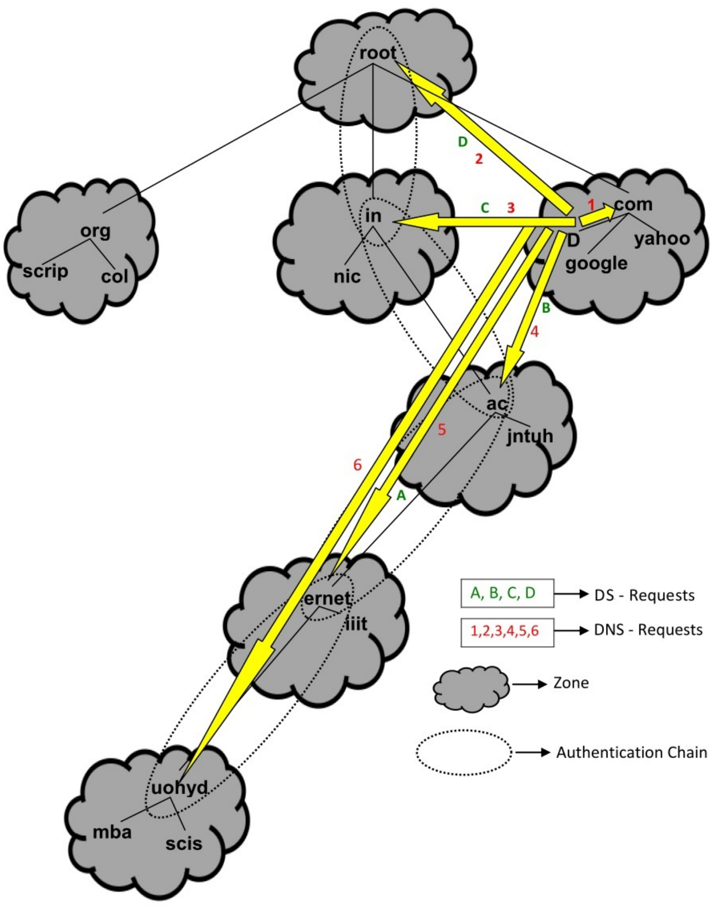


Figure 5.5: DNSSEC Within BDI

5.9 Performance Study with Example

Let us assume that there is a system called `D` *under* `com` *domain which wants to find an IP address of* `scis` *under* `uohyd` *domain and also construct the chain of trust between the server from which it has received an IP address of* `scis` *to the root node, before believing and using the IP address of* `scis`*.* The problem given above can be visualized as shown in Figure 5.5. In chapter 3, we have already discussed and shown how the domain name is resolved to obtain the IP address. Here also the same procedure in-terms of BDI architecture is followed. For a given problem we can construct the following *belief-base* and *action-base* to proceed further.

1. believe(root)

Table 5.5: Belief-base of DNSSEC.

1. send_DNS_request(sent_message, src_address, dst_address)
2. receive_DNS_request(received_message, dst_address, src_address)
3. send_DNS_response(sent_message, src_address, dst_address)
4. receive_DNS_response(received_message, dst_address, src_address)
5. send_DS_request(sent_message, src_address, dst_address)
6. receive_DS_request(received_message, dst_address, src_address)
7. send_DS_response(sent_message, src_address, dst_address)
8. receive_DS_response(received_message, dst_address, src_address)

Table 5.6: Action-base of DNSSEC.

In order to keep the solution simple, we are not including the detailed formats of queries and responses that the actual DNS and DNSSEC systems use. The interpretation of the expressions in the belief-base and the action-base are as follows: To show that the system *D* believes in the *root* node of the DNS name space (which is a DNS tree as shown in figure 5.5) we use the expression *believe(root)* in the belief-base. The sending expression *send-DNS-req(smsg, src, dst)* in the action-base can be interpreted as `the source node (src) sends a message (smsg) to the destination node (dst)`. In a similar manner, we can also interpret other sending expressions *send-DNS-res (smsg, src, dst)*, *send-DS-req(smsg, src, dst)* and *send-DS-res(smsg, src, dst).* The receiving expression *recv-DNS-req(rmsg, dst, src)* in action-base can be interpreted as `the source node(src) received a message (rmsg) from the destination node (dst)`, and similarly we can also understand the expressions *recv-DNS-res(rmsg, dst, src)*, *recv-DS-req(rmsg, dst, src)* and *recv-DS-res(rmsg, dst, src).*

5.9.1 DNS Request Resolution and Verification Process

In the Name-Address resolution process, first the system D sends a DNS-request to the server at *com* domain. The server at "com" domain checks its database and if the requested IP address exists, then it sends that IP address. Otherwise it sends the reference address (that is *root* in our case) as the DNS-response to system D. Now system D checks the DNS-response whether it contains the required IP address or the reference address. If it contains the reference address, then it generates a new DNS-request and sends to the given reference address, else it takes the IP address and stops the process. This process is continued till it receives the required IP address. The part-A of client Algorithm 5.5 and part-A of server Algorithm 5.6 does this task (i.e.DNS request resolution). The client and server algorithms are designed in such a way that they resolve the given DNS query and constructs the chain of trust effectively. Here the client algorithm makes use of the previous results so that it can complete the given task in an efficient manner. The plan or sequence of send and receive actions that the BDI system generates, while resolving the DNS-request *www.scis.uohyd.ac.in.*, is generated from the system D which is under *com* domain, as shown in Figure 5.5.

Once the system D completes the DNS request resolution task successfully, then it starts the construction of chain of trust to verify whether the received IP address is correct or not in-terms of *integrity* and *authentication.* To verify the data integrity and data origin, authentication system D tries to construct the chain of trust between *uohyd* and *root* node as shown in the Figure 5.5. First system D sends a DS-request to the *ernet* domain, which is the parent of *uohyd*

Algorithm 5.5 DNSSEC enabled DNS-Client with BDI

Require: belief base, action base, and FQDN (Fully Qualified Domain Name)
Ensure: IP address, chain of trust

```
    ************* part-A *******************
 1: if is_IP_address_exist(FQDN, belief_base) then
 2:    return IP_address
 3: end if
 4: sent_message ← FQDN
 5: src_address ← sender
 6: dst_address ← parent()
 7: send_DNS_request(sent_message, src_address, dst_address)
 8: receive_DNS_response(received_message, dst_address, src_address)
 9: if is_received_referral(received_message) then
10:    dst_address ← retrieve_referral(received_message)
11:    goto step 7
12: else
13:    IP_address ← retrieve_IP_address(received_message)
14: end if
    ************* part-B *******************
15: if is_source_exist(dst_address, belief_base) then
16:    update_belief_base(IP_address)
17:    Accept and return IP_address
18: else
19:    sent_message ← dst_address
20:    dst_address ← parent(dst_address)
21:    send_DS_request(sent_message, src_address, dst_address)
22:    receive_DS_response(received_message, dst_address, src_address)
23:    if received_message = yes & dst_address $\neq$ root_address then
24:       append_to_list(believed_source_list, dst_address)
25:       goto step 19
26:    else
27:       if received_message = yes & dst_address = root_address then
28:          chain of trust construction successful
29:          update_belief_base(IP_address)
30:          update_belief_base(believed_source_list)
31:          Accept and return IP_address
32:       else
33:          chain of trust construction failed
34:          Reject received IP_address
35:          clear_source_believed_list(believed_source_list)
36:       end if
37:    end if
38: end if
```

domain. The server at *ernet* domain checks its database, and if it finds *uohyd* as a child, then it returns *yes*, otherwise *no* as the DS-response to the system *D*. Now system *D* checks the DS-response and if it contains *yes*, then it generates new DS-request and sends it to the parent of *ernet*. If the DS-response contains *no*, then it stops the process immediately and discards the IP address obtained in DNS resolution process. This process is continued till the *root* node is reached. Once the *root* node is reached, the process is stopped because the *root* node is believed by the system *D* (Part-B of client-Algorithm 5.5 and part-B of server-Algorithm 5.6 does this task (i.e. chain of trust construction)). In Figure 5.5 we have represented the above plan as [1, 2, 3, 4, 5, 6]. In order to keep the diagram simple, we have combined four actions in the above plan into a single action while representing in the figure. Number 1 in the Figure 5.5 represents 1 to 4 actions of Table 5.7. Others can be interpreted in a similar similar way.

Algorithm 5.6 DNSSEC enabled DNS-Server with BDI

```
Require: domain name, belief base, action base
Ensure: IP address, referral, child registration
    ************** part-A *********************
 1: receive_DNS_response(received_message, dst_address, src_address)
 2: if is_in_child_list(received_message) then
 3:     sent_message ← retrieve_IP_address(received_message)
 4: else
 5:     sent_message ← retrieve_referral(received_message)
 6: end if
 7: send_DNS_response(sent_message, src_address, dst_address)
    ************** part-B ********************
 8: receive_DS_request(received_message, dst_address, src_address)
 9: if is_child_registered(received_message) then
10:     sent_message ← yes
11: else
12:     sent_message ← no
13: end if
14: send_DS_response(sent_message, src_address, dst_address)
```

1. send_DNS_request(scis, D, com)
2. receive_DNS_request(scis, D, com)
3. send_DNS_response(root, com, D)
4. receive_DNS_response(root, com, D)
5. send_DNS_request(scis, D, root)
6. receive_DNS_response(scis, D, root)
7. send_DNS_response(in, root, D)
8. receive_DNS_response(in, root, D)
9. send_DNS_request(scis, D, in)
10. receive_DNS_request(scis, D, in)
11. send_DNS_response(ac, in, D)
12. receive_DNS_response(ac, in, D)
13. send_DNS_request(scis, D, ac)
14. receive_DNS_request(scis, D, ac)
15. send_DNS_response(ernet, ac, D)
16. receive_DNS_response(ernet, ac, D)
17. send_DNS_request(scis, D, ernet)
18. receive_DNS_request(scis, D, ernet)
19. send_DNS_response(uohyd, ernet, D)
20. receive_DNS_response(uohyd, ernet, D)
21. send_DNS_request(scis, D, uohyd)
22. receive_DNS_request(scis, D, uohyd)
23. send_DNS_response(IPadd, uohyd, D)
24. receive_DNS_response(IPadd, uohyd, D)

Table 5.7: Plan generated by BDI to resolve DNS query

5.9.2 Belief Base Updation Process

As the plan shown in Table 5.7 gets executed, the belief-base gets updated as follows: Initially, the belief-base contains *believe(root)*, because by default every node believes the *root* node.

1. believe(root)

Table 5.8: Initial belief-base

1. believe(root)
2. parent(com, root)

Table 5.9: Belief-base after step 1 to 4

After execution of steps 1 to 4, system *D* comes to know that *root* is the parent of *com* domain. The axiom *parent(com, root)* can be interpreted as parent of *com* domain is *root*. Similarly after execution of steps 5 to 8, system *D* comes to know that *root* is the parent of *in* domain.

1. believe(root)
2. parent(com, root)
3. parent(in, root)

Table 5.10: Belief-base after step 5 to 8

1. believe(root)
2. parent(com, root)
3. parent(in, root)
4. parent(ac, in)

Table 5.11: Belief-base after step 9 to 12

Execution of steps 9 to 12 will make the system *D* come to know that *in* is the parent of *ac* domain. In a similar manner execution of steps 13 to 16, makes the system *D* realise that *ac* is the parent of *ernet* domain.

1. believe(root)
2. parent(com, root)
3. parent(in, root)
4. parent(ac, in)
5. parent(ernet, ac)

Table 5.12: Belief-base after step 13 to 16

1. believe(root)
2. parent(com, root)
3. parent(in, root)
4. parent(ac, in)
5. parent(ernet, ac)
6. parent(uohyd, ernet)

Table 5.13: Belief-base after step 17 to 20

After execution of steps 21 to 24, system *D* comes to know that *uohyd* knows the IP address of *scis*.

1. believe(root)
2. parent(com, root)
3. parent(in, root)
4. parent(ac, in)
5. parent(ernet, ac)
6. parent(uohyd, ernet)
7. IP address(scis, ”xx.xx.xx.xx”)

Table 5.14: Belief-base after step 21 to 24

5.9.3 Plan Generation Process to Construct Chain of Trust

One of the plans generated by part-B of client Algorithm 5.5 and part-B of server Algorithm 5.6 to construct the ’chain of trust’, from the *uohyd* domain to the *root* domain is as given below:

1. send_DS_request(uohyd, D, ernet)
2. receive_DS_request(uohyd, D, ernet)
3. send_DS_response(yes, ernet, D)
4. receive_DS_response(yes, ernet, D)
5. send_DS_request(ernet, D, ac)
6. receive_DS_request(ernet, D, ac)
7. send_DS_response(yes, ac, D)
8. receive_DS_response(yes, ac, D)
9. send_DS_request(ac, D, in)
10. receive_DS_request(ac, D, in)
11. send_DS_response(yes, in, D)
12. receive_DS_response(yes, in, D)
13. send_DS_request(in, D, root)
14. receive_DS_request(in, D, root)
15. send_DS_response(yes, root, D)
16. receive_DS_response(yes, root, D)

Table 5.15: Plan generated by BDI to construct chain of trust

We can also see this plan in Figure 5.5, which was represented as [A, B, C, D]. Once again to keep the diagram simple we have combined the four actions in the above plan into a single action while representing it in the form of a diagram. The four actions [1 to 4] of Table 5.15 has been coalesced into a single action *A* in Figure 5.5 and the othe part of the figure can also be interpreted in a similar way. As the plan shown above gets executed, the belief-base gets updated as in Table 5.16.

1. believe(root)
2. parent(com, root)
3. parent(in, root)
4. parent(ac, in)
5. parent(ernet, ac)
6. parent(uohyd, ernet)
7. believe IP address(scis)
8. believe(in)
9. believe(ac)
10. believe (ernet)
11. believe(uohyd)

Table 5.16: Belief-base after execution of chain of trust plan

As we can easily observe from the Table 5.16, the four expressions *believe(in)*, *believe(ernet) believe(ac)* and *believe(uohyd)* get appended into the belief-base after successful construction of chain of trust. We can keep the beliefs in the belief-base for a specified time period. If we get a DNS request within a short time, for any system which is under *in* domain or *ac* domain or *ernet* domain or *uohyd* domain, then we need not repeat the whole process of the chain of trust construction to believe the response. Avoiding unnecessary chain of trust constructions will definitely improve the performance of DNSSEC protocol. This will also reduce the response time and Internet traffic.

5.10 Reasoning about DNSSEC with temporal BDI in (Alice - Bob) notation

1. $C- > S : DNS\_req$
2. $S- > C : DNS\_res, RRSIG[DNS\_res], DNSKEY[S]$
3. C "$verify$" $if\ H(DNS\_res) == \{RRSIG[DNS\_res]\}_{DNSKY[S]}\ then$
   $CLR < -cacheLookup(DNSKEY[S])\ goto\ step 4.$
   $else\ discard\ DNS\_res\ and\ exit.$
4. C "$verify$" $if\ CLR =' yes'\ then$
   $Accept\ DNS\_res\ \&\ exit$
5. $C- > R : DS\_req$
6. $R- > C : DS\_res, RRSIG[DS\_res], DNSKEY[R]$
7. C "$verify$" $if\ H(DS\_res) == \{RRSIG[DS\_res]\}_{DNSKEY[R]}\ then\ goto\ step\ 8.$
   $else\ discard\ DS\_res\ and\ exit$
8. C "$verify$" $if\ H(DNSKEY[S]) == (DS\_res)\ then$
   $accept\ DNS\_res,\ write\ DNSKEY[S]\ into\ cache\ and\ exit.$
   $else\ discard\ DNS\_res\ and\ exit.$

5.10.1 The Reasoning Process

- To achieve Origin Authentication we should derive the following rule at the client side **C believes DNSKEY[S]**
- To achieve Integrity we should derive the following rule at the client side **C believes RRSet[DNS_req]**
- A client believing zone's DNSKEY as public key for its zone can be represented as *C believes* $\overset{DNSSKEY}{\longmapsto}$ *ZONE* OR *C believes DNSKEY*
- A client comparing the calculated hash value of the response (DNS_res / DS_res) and encrypted RRSIG with DNSKEY can be represented as *Equal(hash(response),* $\{RRSIG[response]\}_{DNSKEY}$*)*
- Initial assumptions
  A1: *C believes DNSKEY[R]*
- Protocol Annotation
  A2: *C received* $DNS\_res, RRSIG[DNS\_res], DNSKEY[S]$
  A3: *C received* $DS\_res, RRSIG[DS\_res], DNSKEY[R]$
- Protocol comprehended messages
  A4: *C sees* $DNS\_res, RRSIG[DNS\_res], DNSKEY[S]$
  A5: *C sees* $DS\_res, RRSIG[DS\_res], DNSKEY[R]$
- Protocol interpreted messages
  A6: *C sees hash(*DNS_res*)*
  A7: *C sees hash(*DS_res*)*
  A8: *C sees* $\{RRSIG[DS\_res]\}_{DNSKEY[R]}$
  A9: *C sees* $\{RRSIG[DNS\_res]\}_{DNSKEY[S]}$
  A10: *C sees hash(DNSKEY[R])*
  A11: *C sees hash(DNSKEY[S])*
- Deriving protocol goals
  Origin Authentication

A12: $IsInCache(DNSKEY[S])$

A13: **C believes DNSKEY[S]**

From A7 and A8

A14: $Equal(hash(DS\_res),\ \{RRSIG[DS\_res]\}_{DNSKEY[R]})$

From A1 and A14

A15: *C believes DS_res*

From A11 and A15

A16: $hash(DNSKEY[S])\ \equiv\ DS\_res$

From A15 and A16

A17: **C believes DNSKEY[S]**

A18: $writeIntoCache(DNSKEY[S])$

Data integrity

From A6 and A9

A19: $Equal(hash(DNS\_res),\ \{RRSIG[DNS\_res]\}_{DNSKEY[S]})$

From A17 and A19

A17: **C believes DNS_res**

5.11 Results

The following results are obtained using DNSViz and DNS speed test tools. DNSViz is a tool suite for analysis and visualization of Domain Name System (DNS) behavior, including its security extensions (DNSSEC). Using this tool one can visualize the status of a DNS zone. It was designed as a resource for understanding and troubleshooting deployment of the DNS Security Extensions (DNSSEC). It provides a visual analysis of the DNSSEC authentication chain for a domain name and its resolution path in the DNS namespace, and it lists configuration errors detected by the tool. The *DNS Hosting Speed Tool* is used to find the response time of authoritative DNS servers for a domain or host. Using this tool we can also find the response time of Top Level Domain (TLD), and Root DNS servers. The tool provides the information, using which the user can identify the reason and/

or location that causes for DNS hosting performance delays that occurs during resolving of a domain. The tool is also useful for evaluating the DNS performance of service providers and general troubleshooting performance of a domain.

	DNS	DNSSEC
www.nic.in	85	496
www.uohyd.ac.in	186	440
www.jntuh.ac.in	192	380
www.iith.ac.in	163	347
www.svuniversity.edu.in	30	315
www.andhrauniversity.edu.in	48	482

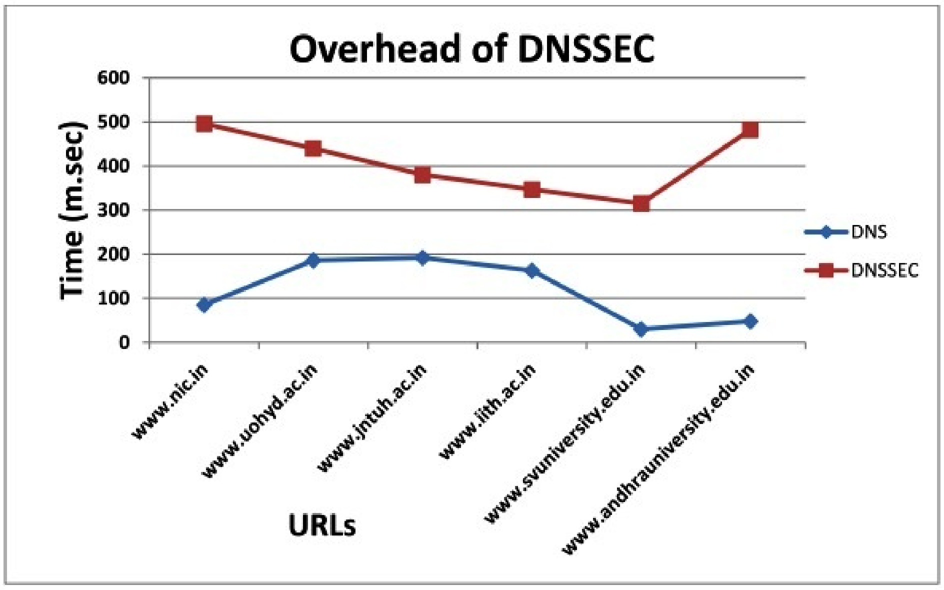


Figure 5.6: DNSSEC overhead

Figure 5.6 shows the overhead of DNSSEC on present DNS system in terms of query resolution time. It is very clear that DNS takes less time compared to DNSSEC to resolve the query. This is because DNSSEC has to construct additional chain of trust to ensure authentication and integrity of the received data.

	DNSSEC	DNSSEC with Temporal BDI
www.jntuh.ac.in	293	293
www.uohyd.ac.in	307	81
www.scis.uohyd.ac.in	307	0
www.acad.uohyd.ac.in	307	0
www.svuniversity.edu.in	222	176
www.andhrauniversity.edu.in	225	71
www.upsc.gov.in	197	150
www.apsc.gov.in	190	71
www.nic.in	150	103

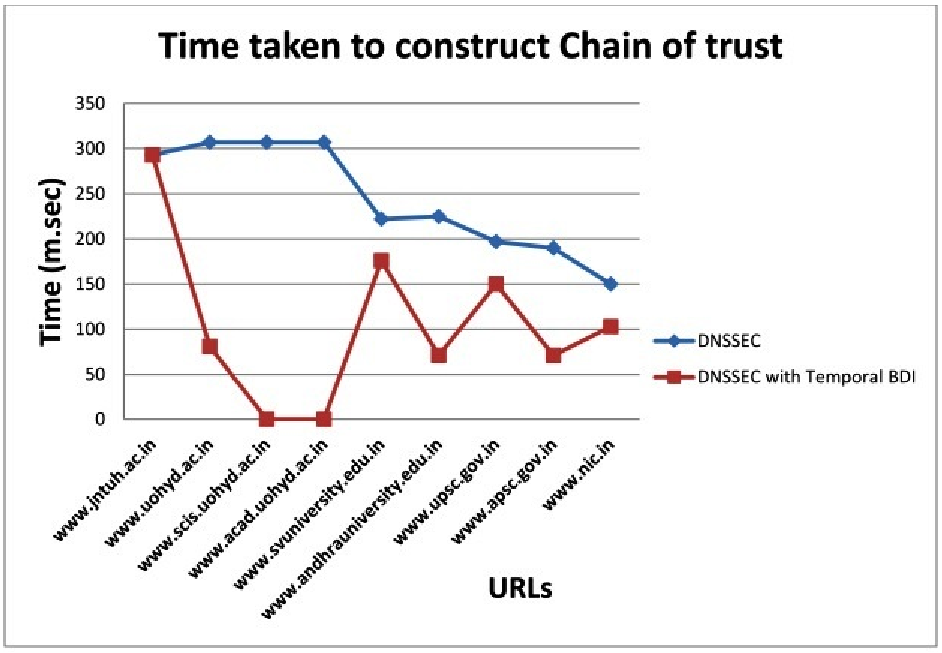
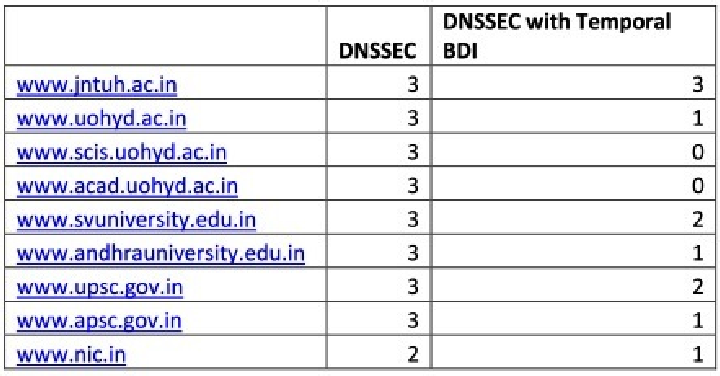


Figure 5.7: Chain of trust construction time

Figure 5.7 shows the comparison of time taken to construct chain of trust between *DNSSEC* and *DNSSEC with Temporal BDI*. It is apparent from the graph that the proposed approach *DNSSEC with Temporal BDI* takes less or equal time compared to original DNSSEC to construct the chain of trust. Here we are making an assumption that if the information related to query is available in the local cache memory then it takes very less time to compute the construction of a chain of trust.



	DNSSEC	DNSSEC with Temporal BDI
www.jntuh.ac.in	3	3
www.uohyd.ac.in	3	1
www.scis.uohyd.ac.in	3	0
www.acad.uohyd.ac.in	3	0
www.svuniversity.edu.in	3	2
www.andhrauniversity.edu.in	3	1
www.upsc.gov.in	3	2
www.apsc.gov.in	3	1
www.nic.in	2	1

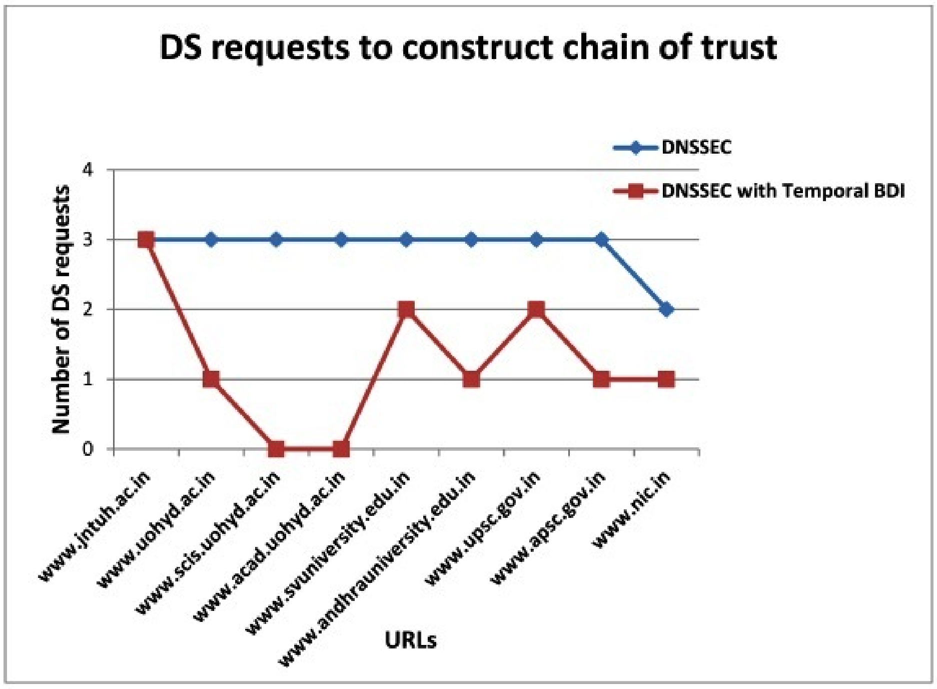


Figure 5.8: Number of DS requests to construct chain of trust

Figure 5.8 shows the number of DS requests required to construct the chain of trust. It is obvious from the graph that the proposed approach *DNSSEC with Temporal BDI* requires less number of DS requests compared to the original DNSSEC. Here once again the assumption is that if the information related to a query is available in the local cache memory then the number of DS requests that needs to be sent is zero. Figures 5.9 and 5.10 represent DNSSEC authentication chain constructions from root to various authenticated domain servers.

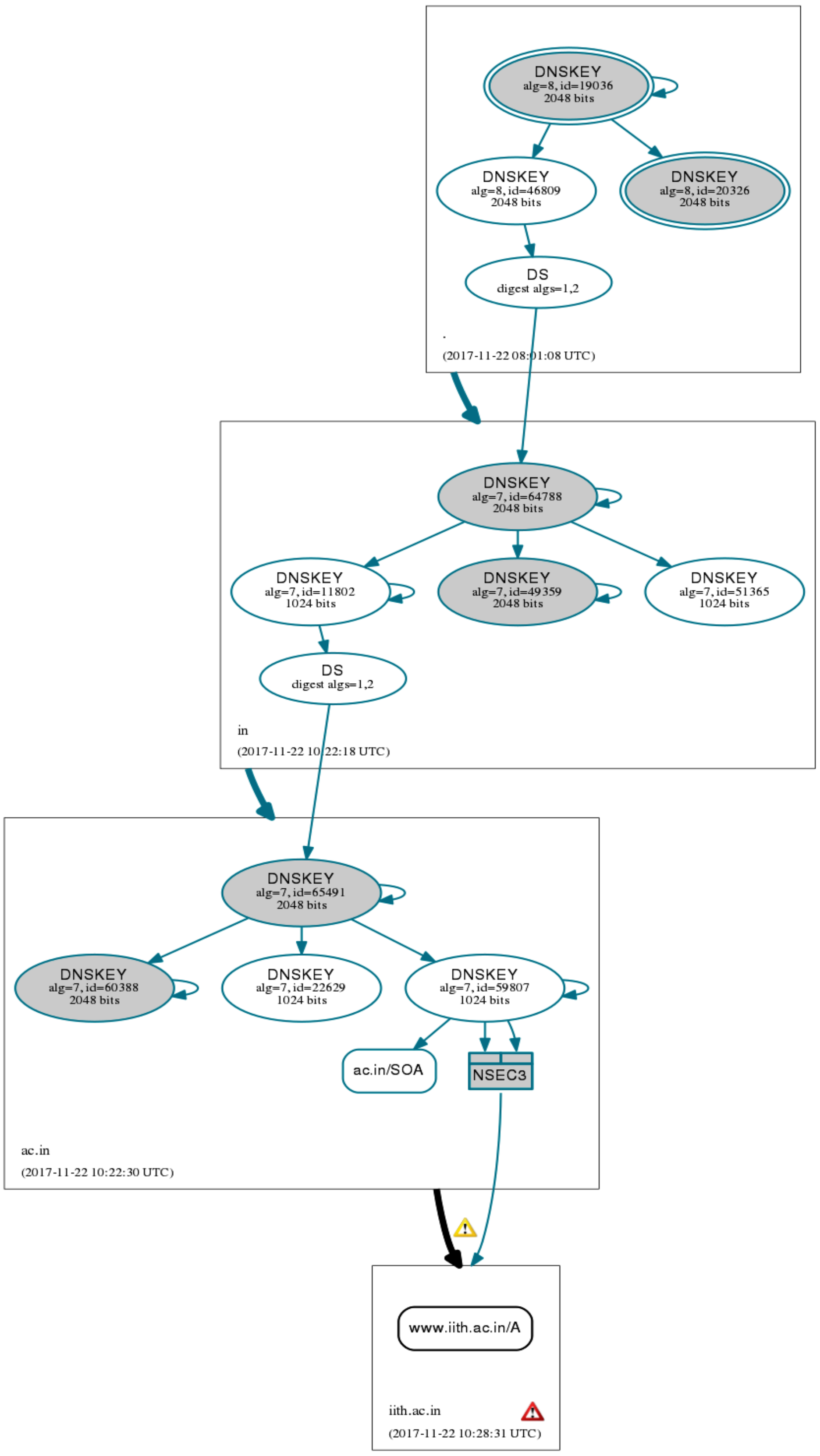


Figure 5.9: DNSSEC Authentication Chain for www.iith.ac.in

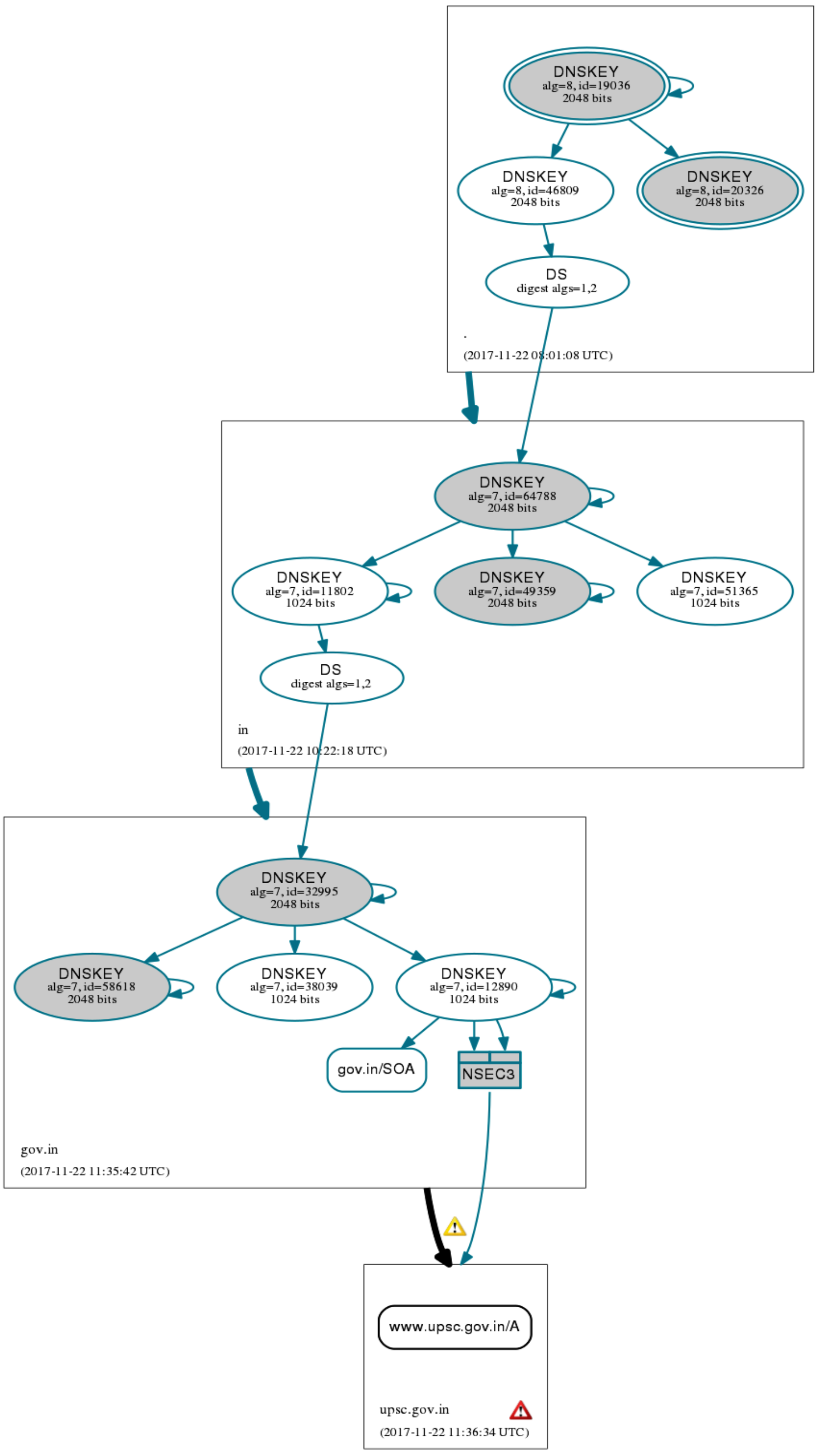


Figure 5.10: DNSSEC Authentication Chain for www.upsc.gov.in

5.12 Summary

The main motivation for writing this chapter is twofold: The first motivation was to outline how BDI-like Agent systems should handle plan failures in an efficient manner by generating plans when *relevant-plans* are not found in the plan-library. Most of the planners like *SAPA* [36, 37] and *Metric-FF* [48] uses `relaxed planning graph` approach to find the heuristics. AgentSpeak(PL) [59] uses GraphPlan approach to generate *context* of a plan in BDI like environment. The major difference between our work and the ones mentioned above is that our GeneratePlan method uses a set of plans (which could be partially instantiated) along with a set of facts, actions and goals to generate a new plan. The languages CanPlan1 [89, 77] and CanPlan2 [76] give an elaborate account of handling goals in the context of plan failures in a BDI environment but the difference with our approach is that they deal with non-temporal goals and plans. Similarly [81, 82] also talk about plan failure but again do not take into account the temporal aspect. To show the feasibility of our method we implemented our planning algorithms on a scenario like travel planning.

The second motivation for writing this chapter was to explain the process of reasoning about DNSSEC from an Agent-Oriented perspective. The main intuition for doing this was to improve the DNSSEC performance in terms of construction of chain of trust. To this end a formal analysis of DNSSEC was carried out in the backdrop of a particular agent-based model called the BDI. For a formal analysis of DNSSEC within BDI, we first developed a belief-base, action-base and plan-library. Then we designed algorithms to accomplish the name-address resolution and the chain of trust construction tasks. The algorithms were designed in such a way that they take the strategic advantage from the results obtained from the previous protocol runs, so that the overall performance of the system is improved.

According to the proposed model, first the DNS protocol will resolve the given DNS query by accessing many name servers in the DNS name space. Once the required response (usually the IP address for a given domain name) is received, it is the responsibility of the DNSSEC protocol to ensure the *integrity* and *authentication* of the received response. As mentioned earlier, the DNSSEC protocol makes use of the concept called *chain of trust* to ensure the integrity and authentication of the received response. According to the concept of *locality of reference*, most of the time we retrieve the data or we search for data in the nearby locations with reference to present location. We have used this concept in improving the performance of the overall system. In the present scenario, the DNS system saves the DNS responses (usually the IP address of a given domain) in the cache memory, to make use of them at a later point of time when similar requests are received. By doing so, the DNS system could improve the performance by avoiding the execution of domain resolving procedure. Here we applied similar technique to that of DNSSEC protocol within BDI system to improve the performance. The idea is to save the chain of trusts, that are constructed successfully in-terms of believed-servers list. By doing so were able to avoid the reconstruction of chain of trust to the servers that are present in the believed-servers list. This can avoid unnecessary chain of trust constructions which will improve the performance of DNSSEC as well as reduce the response time and network traffic.

Chapter 6

Conclusions and Future Work

In this thesis we tried to address three research issues related to the Domain Name System Security (DNSSEC) protocol. The first issue was concerned with the formal analysis of DNSSEC wherein we used a logic based frame work called SVO to formally represent and reason about DNSSEC. We gave an in-depth analysis of DNSSEC to start with and used SVO logic for formally representing DNSSEC and also specifying the nuances involved in DNSSEC. Thereafter we proposed an extension of DNSSEC wherein two new axioms were added to the original SVO logic. Formal Analysis of DNSSEC in the new set up was carried out by verifying origin authentication and Data integrity with the help of message passing examples. By making use of inference rules as well as axioms of SVO logic we were able to successfully derive proof for Data origin authentication and Data integrity of DNSSEC protocol. Many formal approaches for specifying and verifying security protocols are available in the literature but to the best of our knowledge our work on *Reasoning about DNSSEC* [13] was the first work that addressed the issue of specification and verification of the DNSSEC protocol in a logical framework. Later on other formal frameworks [42] have also been proposed.

The second research issue we addressed in this thesis is related to the validation of DNSSEC protocol using an automatic validation tool like AVISPA. We demonstrated how the AVISPA tool can be used to validate the DNSSEC protocol. We have represented DNSSEC protocol in HLPSL which is later translated into IF Form by HLPSLIF converter. This HLPSL file is fed to back ends of the AVISPA tool to obtain the status of the protocol. The IF form of the protocol is, then analyzed using four backend analyzers called OFMC, SATMC, CL-AtSe and TA4SP to check the correctness of the given protocol. It gives status like SAFE or UNSAFE which tells us whether an intruder attack of the protocol has happened or not. For a simple protocol like Alice Bob we got results as SAFE which means that there was no intruder attacks. The simulation results ensured that the DNSSEC protocol is safe which means that no attacks are possible on specified goals. We have also displayed the results pertaining to single as well as multiple sessions, with different backend analysers. Finally we concluded that the DNSSEC protocol is successfully validated as no attacks were found against it's authentication and integrity goals.

The third research issue was related to the performance enhancement of the DNSSEC protocol. To achieve this end what we did was to explain the process of reasoning about DNSSEC from an Agent-Oriented perspective. To this end a formal analysis of DNSSEC was proposed to carry out in the backdrop of a particular agent-based model called the BDI. The main intuition for doing this was to improve the DNSSEC performance in terms of construction of chain of trust. A temporal planning model for BDI was developed which included a belief-base, action-base and plan-library. For the implementation of temporal planning within BDI we designed algorithms to find the plans for a given goal by making use of the plan library. A categorisation of plans into *relevant*, *applicable* and *extended applicable* plans were made to differentiate the planning at different time steps and a plan which had lesser time than the given goal time was chosen as the final plan. After implementing temporal planning in BDI successfully, we utilized this technique to improve the performance of DNSSEC protocol wherein we further designed algorithms to accomplish the name-address resolution and the chain of trust construction tasks. The algorithms were designed in such a way that they take the strategic advantage from the results obtained from the previous protocol runs, so that the overall performance of the system is improved.

This improvement will also reduce the Internet traffic by avoiding unnecessary chain of trust constructions. This will also improve response time of DNSSEC protocol. BDI system uses the saved authentication results to answer similar queries in future within the valid time period. The performance of DNSSEC protocol is improved by effectively utilizing the results of previous chain of trust constructions.

In this work we have concentrated mostly on syntactic representation of the DNSSEC protocol and have provided a method for reasoning about it in the background of a logical framework. For complete formal analysis of a protocol, one needs to develop both syntactic as well as semantic representation. As part of my future work, I want to concentrate on semantic representation and reasoning of DNSSEC protocol as well as other protocols. I also want to concentrate on reasoning about various kinds of attacks on the DNS protocol. We would also like to apply the formal apparatus built in this thesis on other security protocols in various domains like cloud computing, peer to peer networks, wireless sensor networks etc.

References

[1] Martín Abadi. Secrecy by typing in security protocols. *Journal of the ACM (JACM)*, 46(5):749–786, 1999.

[2] Martin Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 20(3):395, 2007.

[3] Martin 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, 1991.

[4] Roberto M Amadio and Witold Charatonik. On name generation and set-based analysis in the dolev-yao model. In *CONCUR*, volume 2, pages 499–514. Springer, 2002.

[5] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. Dns security introduction and requirements. RFC 4033, Internet Engineering Task Force, 2005.

[6] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. Resource records for the dns security extensions. RFC 4034, Internet Engineering Task Force, 2005.

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

[8] Alessandro Armando and Luca Compagna. Satmc: a sat-based model checker for security protocols. In *JELIA*, volume 3229, pages 730–733. Springer, 2004.

[9] Alessandro Armando, David Basin, Yohan Boichut, Yannick Chevalier, Luca Compagna, Jorge Cuéllar, Hankes P Drielsma, Pierre-Cyrille Héam, Olga Kouchnarenko, Jacopo Mantovani, et al. The avispa tool for the automated validation of internet security protocols and applications. In *International conference on computer aided verification*, pages 281–285. Springer, 2005.

[10] Giuseppe Ateniese and Stefan Mangard. A new approach to dns security (dnssec). In *Proceedings of the 8th ACM Conference on Computer and Communications Security*, pages 86–95. ACM, 2001.

[11] Team AVISPA. Hlpsl tutorial, 2014.

[12] Franz Baader and Klaus U Schulz. Unification in the union of disjoint equational theories: Combining decision procedures. *Journal of Symbolic Computation*, 21(2):211–243, 1996.

[13] Kollapalli Ramesh Babu, Vineet Padmanabhan, and Wilson Naik Bhukya. Reasoning about DNSSEC. In *Multi-disciplinary Trends in Artificial Intelligence - 5th International Workshop, MIWAI 2011*, pages 75–86, 2011.

[14] D Basin and G Denker. Maude versus haskell: A comparison in security protocol analysis. In *International Workshop on Rewriting Logic and Applications, Electronic Lecture Notes on Theoretical Computer Science*, volume 36, 2001.

[15] David Basin. Lazy infinite-state analysis of security protocols. In *Secure NetworkingCQRE [Secure]99*, pages 30–42. Springer, 1999.

[16] David Basin, Sebastian Mödersheim, and Luca Vigano. Ofmc: A symbolic model checker for security protocols. *International Journal of Information Security*, 4(3):181–208, 2005.

[17] Jason Bau and John C. Mitchell. A security evaluation of DNSSEC with NSEC3. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2010, San Diego, California, USA, 28th February - 3rd March 2010*, 2010.

[18] Steven M. Bellovin. Using the domain name system for system break-ins. In *Proceedings of the 5th USENIX Security Symposium, Salt Lake City, Utah, USA, June 5-7, 1995*, 1995.

[19] Bruno Blanchet and Andreas Podelski. Verification of cryptographic protocols: Tagging enforces termination. In *FOSSACS'03/ETAPS'03*, pages 136–152. Springer, 2003.

[20] Yohan Boichut, Pierre-Cyrille Héam, and Olga Kouchnarenko. Automatic verification of security protocols using approximations. Technical report, INRIA, 2006.

[21] Rafael H Bordini, Ana LC Bazzan, Rafael de O Jannone, Daniel M Basso, Rosa M Vicari, and Victor R Lesser. Agentspeak (xl): Efficient intention selection in bdi agents via decision-theoretic task scheduling. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 3*, pages 1294–1302. ACM, 2002.

[22] Michael E. Bratman. *Intention, Plans and Practical Reason.* Cambridge University Press, 1987.

[23] Jan M. Broersen, Mehdi Dastani, Joris Hulstijn, Zhisheng Huang, and Leendert W. N. van der Torre. The BOID architecture: conflicts between beliefs, obligations, intentions and desires. In *Agents*, pages 9–16, 2001.

[24] Michael Burrows, Martin Abadi, and Roger M Needham. A logic of authentication. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, volume 426, pages 233–271. The Royal Society, 1989.

[25] Frederick Butler, Iliano Cervesato, Aaron D. Jaggard, and Andre Scedrov. A formal analysis of some properties of kerberos 5 using MSR. In *15th IEEE Computer Security Foundations Workshop (CSFW-15 2002), 24-26 June 2002, Cape Breton, Nova Scotia, Canada*, page 175, 2002.

[26] Levente Buttyán and Hubaux Jean-Pierre. Rational exchange-a formal model based on game theory. *Electronic Commerce*, pages 114–126, 2001.

[27] Yannick Chevalier and Laurent Vigneron. A tool for lazy verification of security protocols. In *Automated Software Engineering, 2001.(ASE 2001). Proceedings. 16th Annual International Conference on*, pages 373–376. IEEE, 2001.

[28] Yannick Chevalier, Luca Compagna, Jorge Cuellar, Paul Hankes Drielsma, Jacopo Mantovani, Sebastian Mödersheim, and Laurent Vigneron. A high level protocol specification language for industrial security-sensitive protocols. In *Workshop on Specification and Automated Processing of Security Requirements-SAPS'2004*, pages 13–p. Austrian Computer Society, 2004.

[29] Edmund M. Clarke and Jeannette M. Wing. Formal methods: State of the art and future directions. *ACM Comput. Surv.*, 28(4):626–643, 1996.

[30] Alexander Cowperthwaite. Trust Models for Remote Hosts. Master's thesis, Computer Science, Carleton University, Canada, 2011.

[31] Cas Cremers. Feasibility of multi-protocol attacks. In *Availability, Reliability and Security, 2006. ARES 2006. The First International Conference on*, pages 8–pp. IEEE, 2006.

[32] Cas J. F. Cremers. *The Scyther Tool: Verification, Falsification, and Analysis of Security Protocols*, pages 414–418. Springer Berlin Heidelberg, 2008.

[33] Cas JF Cremers. The scyther tool: Verification, falsification, and analysis of security protocols. In *CAV*, volume 8, pages 414–418. Springer, 2008.

[34] Lavindra De Silva, Sebastian Sardina, and Lin Padgham. First principles planning in bdi systems. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 1105–1112. ACM, 2009.

[35] Mark d'Inverno and Michael Luck. Engineering agentspeak(l): A formal computational model. *Journal of Logic & Computation*, 8(3):233–260, 1998.

[36] Minh Do and Subbarao Kambhampati. Sapa: A domain-independent heuristic metric temporal planner. In *Sixth European Conference on Planning*, pages 57–68. AAAI Press, 2014.

[37] Minh Binh Do and Subbarao Kambhampati. Planning graph-based heuristics for cost-sensitive temporal planning. In *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems, April 23-27, 2002, Toulouse, France*, pages 3–12, 2002.

[38] Danny Dolev and Andrew Yao. On the security of public key protocols. *IEEE Transactions on information theory*, 29(2):198–208, 1983.

[39] Danny Dolev, Shimon Even, and Richard M Karp. On the security of ping-pong protocols. *Information and Control*, 55(1-3):57–68, 1982.

[40] D. Eastlake and C. kaufman. Domain name system security extensions. rfc 2065. https://tools.ietf.org/html/rfc2065, 1997.

[41] D. Eastlake and C. kaufman. Domain name system security extensions. rfc 2535. https://tools.ietf.org/html/rfc2535, 1999.

[42] Ezequiel Bazan Eixarch, Gustavo Betarte, and Carlos Daniel Luna. A formal specification of the DNSSEC model. *ECEASST*, 48, 2011.

[43] Joe Alby et.al. Dnssec root zone high level technical architecture. Technical report, Root DNSSEC design Team, 2010.

[44] Boris A. Galitsky. Exhaustive simulation of consecutive mental states of human agents. *Knowl.-Based Syst.*, 43:1–20, 2013.

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

[46] Gilles Guette. Key revocation system for DNSSEC. *JNW*, 3(6):54–61, 2008.

[47] Amir Herzberg and Haya Shulman. Negotiating dnssec algorithms over legacy proxies. In *International Conference on Cryptology and Network Security*, pages 111–126. Springer, 2014.

[48] Jörg Hoffmann. The metric-ff planning system: Translating"ignoring delete lists"to numeric state variables. *Journal of Artificial Intelligence Research*, 20:291–341, 2003.

[49] Geoff Huston. Dnssec - the theory, 2006.

[50] Yong Jin, Masahiko Tomoishi, and Nariyoshi Yamai. An advanced client based dnssec validation and preliminary evaluations toward realization. In *Information and Communication Technology Convergence (ICTC), 2016 International Conference on*, pages 178–183. IEEE, 2016.

[51] Olaf Kolkman. Dnssec howto, a tutorial in disguise. http://www.nlnetlabs.nl/dnssec-howto/dnssec-howto.pdf, 2009.

[52] Dennis Longley and Simon Rigby. An automatic search for security flaws in key management schemes. *Computers & Security*, 11(1):75–89, 1992.

[53] Gavin Lowe. An attack on the needham-schroeder public-key authentication protocol. *Inf. Process. Lett.*, 56(3):131–133, 1995.

[54] Gavin Lowe. Breaking and fixing the needham-schroeder public-key protocol using fdr. In *Proceedings of the Second International Workshop on Tools*

and Algorithms for Construction and Analysis of Systems, pages 147–166. Springer, 1996.

[55] Gavin Lowe. Casper: A compiler for the analysis of security protocols. *Journal of computer security*, 6(1-2):53–84, 1998.

[56] Ji Ma, Mehmet A. Orgun, and Abdul Sattar. Analysis of authentication protocols in agent-based systems using labeled tableaux. *IEEE Trans. Systems, Man, and Cybernetics, Part B*, 39(4):889–900, 2009.

[57] C. Meadows. A procedure for verifying security against type confusion attacks. In *16th IEEE Computer Security Foundations Workshop, 2003. Proceedings.*, pages 62–72. IEEE, 2003.

[58] Catherine Meadows. The nrl protocol analyzer: An overview. *The Journal of Logic Programming*, 26(2):113–131, 1996.

[59] Felipe Meneguzzi and Michael Luck. Leveraging new plans in agentspeak (pl). In *International Workshop on Declarative Agent Languages and Technologies*, pages 111–127. Springer, 2008.

[60] Felipe R Meneguzzi, Avelino F Zorzo, and Michael C Móra. Mapping mental states into propositional planning. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-agent Systems-Volume 3*, pages 1514–1515. IEEE Computer Society, 2004.

[61] Felipe Rech Meneguzzi, Avelino Francisco Zorzo, and Michael da Costa Móra. Propositional planning in bdi agents. In *Proceedings of the 2004 ACM symposium on Applied computing*, pages 58–63. ACM, 2004.

[62] Daniel Migault, Cédric Girard, and Maryline Laurent. A performance view on dnssec migration. In *Network and Service Management (CNSM), 2010 International Conference on*, pages 469–474. IEEE, 2010.

[63] Daniel Migault, Stanislas Francfort, Stéphane Sénécal, Emmanuel Herbert, and Maryline Laurent. Overcoming dnssec performance issues with dht-based architectures. In *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, pages 816–819. IEEE, 2013.

[64] Daniel Migault, Stephane Senecal, Stanislas Francfort, Emmanuel Herbert, and Maryline Laurent. Prefetching to overcome dnssec performance issue on large resolving platform. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2013 12th IEEE International Conference on*, pages 694–703. IEEE, 2013.

[65] 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.

[66] Paul V. Mockapetris. Domain names: Implementation specification. RFC 1035, Internet Engineering Task Force, 1983.

[67] Paul V. Mockapetris. Domain names-concepts and facilities. RFC 1034, Internet Engineering Task Force, 1987.

[68] Paul V. Mockapetris and Kevin J. Dunlap. Development of the domain name system. *SIGCOMM Comput. Commun. Rev.*, 18(4):123–133, 1988.

[69] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21(12):993–999, 1978.

[70] Dan M. Nessett. A critique of the burrows, abadi and needham logic. *SIGOPS Oper. Syst. Rev.*, 24(2):35–38, 1990.

[71] Mehmet A. Orgun, Guido Governatori, and Chuchang Liu. Modal tableaux for verifying stream authentication protocols. *Autonomous Agents and Multi-Agent Systems*, 19(1):53–75, 2009.

[72] Anand S Rao. Agentspeak (l): Bdi agents speak out in a logical computable language. In *European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, pages 42–55. Springer, 1996.

[73] AW Roscoe, JCP Woodcock, and Lars Wulf. Non-interference through determinism. In *European Symposium on Research in Computer Security*, pages 31–53. Springer, 1994.

[74] Thijs Rozerkrans and Rene Klomp. Reliable client-server connections. , System & Network Engineering, University of Amsterdam, 2013.

[75] Michaël Rusinowitch and Mathieu Turuani. Protocol insecurity with a finite number of sessions and composed keys is np-complete. *Theoretical Computer Science*, 299(1-3):451–475, 2003.

[76] Sebastian Sardina and Lin Padgham. Goals in the context of bdi plan failure and planning. In *Proceedings of the 6th international joint conference on Autonomous agents and multi-agent systems*, page 7. ACM, 2007.

[77] Sebastian Sardina, Lavindra de Silva, and Lin Padgham. Hierarchical planning in bdi agent programming languages: A formal approach. In *Proceedings of the fifth international joint conference on Autonomous agents and multi-agent systems*, pages 1001–1008. ACM, 2006.

[78] Helmut Seidl and Neeraj Kumar Verma. Flat and one-variable clauses: Complexity of verifying cryptographic protocols with single blind copying. *ACM Trans. Comput. Logic*, 9(4):28:1–28:45, 2008.

[79] Paul F Syverson and Paul C Van Oorschot. On unifying some cryptographic protocol logics. In *Research in Security and Privacy, 1994. Proceedings., 1994 IEEE Computer Society Symposium on*, pages 14–28. IEEE, 1994.

[80] TA Team et al. Avispa v1. 1 user manual. *Information Society Technologies Programme (June 2006), http://avispa-project. org*, 2006.

[81] John Thangarajah, James Harland, David Morley, and Neil Yorke-Smith. Aborting tasks in bdi agents. In *Proceedings of the 6th international joint conference on Autonomous agents and multi-agent systems*, page 6. ACM, 2007.

[82] John Thangarajah, James Harland, David Morley, and Neil Yorke-Smith. Suspending and resuming tasks in bdi agents. In *Proceedings of the 7th international joint conference on Autonomous agents and multi-agent systems-Volume 1*, pages 405–412. ACM, 2008.

[83] Mathieu Turuani. The cl-atse protocol analyser. In *International Conference on Rewriting Techniques and Applications*, pages 277–286. Springer, 2006.

[84] Hans van Ditmarsch, Joseph Y. Halpern, Wiebe van Der Hoek, and Bartled Kooi. *Handbook of Epistemic Logic*. College Publications, 2015.

[85] Paul van Oorschot. Extending cryptographic logics of belief to key agreement protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 232–243. ACM, 1993.

[86] Zheng Wang. Optimizing negative caching for dnssec-oblivious resolvers. In *Network Computing and Applications (NCA), 2015 IEEE 14th International Symposium on*, pages 267–274. IEEE, 2015.

[87] Gerhard Weiss. *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT press, 1999.

[88] Jan Wessels. Applications of ban-logic, 2001.

[89] Michael Winikoff, Lin Padgham, James Harland, and John Thangarajah. Declarative & procedural goals in intelligent agent systems. In *Proceedings of the Eights International Conference on Principles and Knowledge Representation and Reasoning (KR-02)*, pages 470–481. Morgan Kaufmann, 2002.

[90] Michael Wooldridge. *Reasoning About Rational Agents*. MIT Press, 2000.

List of Publications

1. Kollapalli Ramesh Babu, Vineet Padmanabhan and Wilson Naik Bhukya, *Reasoning about DNSSEC*, Fifth Multidisciplinary International Conference on Artificial Intelligence (MIWAI 2011), pages 75 - 86, LNAI, Springer.

2. Vineet Padmanabhan, Abdul Sattar, Guido Governatori, and Kollapalli Ramesh Babu, *Incorporating temporal planning within a BDI architecture*, 5th Indian International Conference on Artificial Intelligence (IICAI 2011) pages 1618 - 1636.

3. Kollapalli Ramesh Babu and Vineet Padmanabhan, *Automated validation of DNSSEC*, Accepted in the International Conference on Computing Analytics and Networking (ICCAN 2017), 15-16 Decmber-2017, Odhisa.

4. Kollapalli Ramesh Babu and Vineet Padmanabhan, *BDI based performance enhancement in DNSSEC*, Accepted in the 14th International conference on Distributed Computing and Internet technology (ICDCIT 2018), 11-13 January-2018, Odhisa.

DNSSEC: Verification, Validation and Proposal for Enhancement

A thesis submitted during 2017 to the University of Hyderabad
in partial fulfillment of the requirements for the award of the Degree

Doctor of Philosophy

in

Computer Science

by

Kollapalli Ramesh Babu

Reg. No. 08MCPC09

School of Computer and Information Sciences

University of Hyderabad

P.O. Central University, Gachibowli

Hyderabad - 500 046, India.

Abstract

It is very common these days for any user of the internet to type a hostname like `uohyd.ac.in` into a web browser like google to retrieve relevant information about the domain from the world wide web. The translation of human readable hostnames such as `uohyd.ac.in` into Internet protocol addresses like 183.82.173.128 is done with the help of Domain Name System (DNS). The Domain Name System has a hierarchical distributed database structure and does not include any security as it was designed to be a public database. Almost in every interaction we have with the internet like Email, Web services etc. we make use of DNS and therefore there is a strong demand for securing the communication within the DNS system. DNSSEC is a security protocol designed to authenticate and protect the integrity of DNS through the use of public key-based digital signatures. Even though the security protocols are designed and implemented with utmost care, it is proved that many of the security protocols which have been in use for many years have logical flaws hidden inside. In this thesis the first issue we address is with regard to providing a formal framework based on SVO logic for representing and reasoning about DNSSEC so as to validate it against any logical flaws. Formal Analysis of DNSSEC was carried out by verifying *origin authentication* and *Data integrity* with the help of message passing examples. By making use of the inference rules as well as the axioms of SVO logic we were able to successfully derive the proof for Data origin authentication and Data integrity of the DNSSEC protocol. The second issue is related to the validation of DNSSEC protocol wherein we had to translate the specification in SVO logic into a machine understandable format and this was made possible through the AVISPA tool and HLPSL converter. The simulation results ensured that the DNSSEC protocol is safe. The third issue is related to the performance enhancement in DNSSEC where the challenge was not only in designing a provable protocol but also to get one that works with *time limits.* We had to adopt an agent-oriented approach to solve this issue and proposed a temporal planning model through which an improvement in the DNSSEC performance in terms of construction of *chain of trust* was achieved.

1 Introduction

In this age of internet, it is well known that, every system that is connected via a computer network is assigned with a unique address called the Internet Protocol Address (IP Address), to identify and access different systems. These addresses are numerical labels assigned to each device connected to a computer network and makes use of the *Internet Protocol* (IP) for communication. Humans make use of these networked systems and find it easy to locate different websites by typing in different domain names rather than the numerical labels. The `domain name system` (DNS) [18, 17, 16] maps the domain names that people have used to locate a website to the IP address of the computer that is used to locate a website. For example, if someone types `uohyd.ac.in` into a web browser like google, a server behind the scenes will map that name to the IP address 206.19.49.149.

The Domain Name System has a distributed database structure and does not include any security as it was designed to be a public database. This has lead to security vulnerabilities wherein the DNS process of *looking a site up on the internet* for the user could be hijacked and the user could be send to the hijacker's own deceptive web site for account and password collection. In other words an attacker can fool a cache into accepting false DNS data. It is also the case that various man-in-the-middle kind of attacks are possible. To overcome these vulnerabilities a technology called DNS Security Extensions (DNSSEC) was introduced that add security to the Domain Name System (DNS) protocol by enabling DNS responses to be validated. With DNSSEC, the DNS protocol is much less susceptible to certain types of attacks, particularly DNS spoofing attacks. The main design motive behind DNSSEC is not to end the various kinds of attacks but to make the attacks detectable by the end-user. This in a way saves users from doing online transactions on the wrong server even if a secured connection is used and the address in the browser looks correct. DNSSEC provides security to DNS system by adding four new resource records (RRs) and *constructing a chain of trust.* DNSSEC protocol provides two types of security services, namely, *data integrity* and *data authentication.* Data integrity ensures that the `received` data is the same as that of the `sent` one. Data authentication, on the other hand, ensures the genuineness of a sender server.

The challenge is not only in designing a provable protocol but also to get one that works in a desired time limit. In order to construct a chain of trust among servers lying between network edge server and a domain server, a client of an edge server sends a request to its immediate parent server and in return the latter sends a similar request to its parent until the request reaches the domain root-server where the communicating client-server genuineness can be established. This process establishes a chain of trust among the servers lying between the communicating clients. Unless this chain of trust is established the issues of authentication and data integrity cannot be resolved. Data received through an unauthenticated communication is discarded. The process of establishing a chain of trust is fully dependent on the DNSSEC protocol.

Like any other protocol, the efficiency of DNSSEC is due to its specification and concreteness. A protocol is a concrete one when its functionalities are provable. In the literature it is often referred to as proving the correctness of a protocol. To achieve the correctness of a protocol, formal analysis of the protocol needs to be carried out. This is the *first issue* this thesis addresses. In this thesis we provide a formal analysis of DNSSEC in the background of Modal Logic which to the best of our knowledge have not been attempted before. Before making a software protocol operational, it is extremely important to verify its specification. In the case of a protocol with a formal specification, its functionalities need to be proved to ensure a desired goal (protocol functionality) is reachable from the given ground truth (logic based specification) of the protocol. In this case, DNSSEC's authenticity as well as data integrity are to be proved from its formal specification. Logic based verification mechanism verifies the given security protocols at a high level of abstraction, though in general attacks on security protocols are nonintuitive and subtle in nature. In order to validate the security functionalities of a given security protocol at low-level of abstraction, we need to take the help of automatic validation tools. Keeping this need in view, the *second issue* addressed in this thesis is about the automatic validation of DNSSEC protocol. In this thesis we make use of the AVISPA tool for validating the DNSSEC protocol. The *third issue* to be addressed of a protocol is its timeliness. Particularly in the case of an authentication protocol like DNSSEC, timeliness is the demanding one because it is not desirable for a user needing Internet services to wait too long for authentication. To this end we adopt an agent-oriented approach to solve this issue and propose a temporal planning model through which an improvement in the DNSSEC performance in terms of construction of chain of trust is achieved. In a nutshell, the three issues addressed in this thesis with regard to DNSSEC protocol are:

1. Formal analysis of DNSSEC protocol using SVO logic framework.
2. Automatic validation of DNSSEC protocol using the AVISPA tool.
3. Performance enhancement of DNSSEC using temporal BDI agent model.

On identification of the above three issues, in the next section we present the research challenges this thesis has taken up.

2 Research Challenges

Designing authentication protocols like DNSSEC is always a challenging task because these protocols are often error prone. The best example of such a flawed protocol is the Needham-Schroeder public key protocol [19] . In this simple public key protocol a flaw was found after seventeen years of its publication by *Lowe* using model checker FDR (Failures Divergences Refinement) [14]. Another *logical flaw* was discovered in the renegotiation feature of the widely used TLS (Transport Layer Security) protocol, thirteen years after the first version of the protocol was published. Several type-confusion attacks against the Group Domain of Interpretation Protocol [15] were identified by using NRL protocol analyzer. In a similar manner, there are number of instances in the literature where logical flaws in security protocols were discovered only after the protocol was in use for many years.

In the previous section we have identified three research issues related to DNSSEC protocol. The *first issue* was related to the formal analysis of the DNSSEC protocol. Formal analysis for security protocols is mainly done to unveil the *logical flaws* and/or to prove correctness of a given protocol. The challenges involved in formal analysis of a security protocol includes that of comprehending a security protocol thoroughly in terms of initial conditions, assumptions, constraints, messages exchanged etc. since these factors play a vital role in deriving the goals of a protocol and proving the correctness of a protocol. Misunderstanding some concept/part of a protocol will definitely lead to wrong conclusions which in turn could result in either proving a correct protocol wrong or a wrong protocol correct. There are many instances in the literature where incorrect protocols are proved formally correct and vice versa. This happens mainly due to wrong interpretation or lack of thorough understanding of a protocol.

The *second issue* we mentioned was related to the development of an automated tool for validation of the DNSSEC protocol. Basically, the security protocols are very complex in nature. Most of the time manual validation of such protocols clearly outpaces human capacity. This is because the vulnerabilities are not directly related to cryptographic techniques implemented by the security protocols. Some of the vulnerabilities are related to bad interactions among multiple parallel sessions of the same protocol and are very subtle in nature. For instance, an intruder can make use of a *protocol agent* as an oracle to obtain some crucial information which the intruder could not generate on its own. The intruder will make use of the information thus obtained to forge new messages and gets it injected into other parallel sessions. To identify these type of attacks we need the assistance of automation tools through which we can test the protocol with more number of parallel sessions and other constraints. Research challenge in this case is writing the protocol specification in a language supported by the tool without any errors which in turn also needs a thorough understanding of the protocol. Another challenge is selecting the most appropriate tool that matches our requirements and should be the best among the existing ones, which can handle the protocol under consideration.

The *third issue* we mentioned in the previous section is with regard to improving the performance of the DNSSEC protocol. DNSSEC protocol protects the DNS system by verifying the authentication and integrity of DNS responses. DNSSEC protocol is able to achieve this authentication and integrity by imposing some overhead on the present DNS system. What these overheads mean from the clients perspective is that of acquiring additional information to authenticate the received data. In order to get additional information, the client has to send more number of queries and receive response from servers as well as perform additional operations like storing, comparing etc. The overhead from servers perspective is in the maintenance of additional information regarding all of its immediate descendant servers which are responsible for sub-domains (zones). The overhead from networks point of view is in the handling of additional traffic due to extra requests/ responses that are sent/ received by clients and servers. Research challenge in this third issue has to do with the minimisation of the overhead wherever it is possible so that the overall performance of the system can be improved. Research challenge in specific is in the reduction of the number of chain of trust constructions as far as possible or else reducing the time to construct the chain of trust in all possible ways. Another challenge is to ensure that the authentication process takes place in a `timed` manner.

3 Proposed Solutions

The first problem that we addressed in this thesis is that of formally analysing the DNSSEC protocol. Formal analysis can be interpreted as the usage of "mathematical techniques" to ensure that the design or implementation of the security protocol confirms to the specified security goals. Mathematical techniques or formal methods that are used to verify security protocols are broadly classified into three categories: The first one is based on *Model checking* methods whereas the second one is based on the method of *Theorem provers*, and the third one is based on methods that make use of *Logical inference* [9]. Methods based on model checking explore all states and transitions in the model by using smart and domain specific techniques. Methods based on Theorem provers search at a higher level of abstraction for chains of logic that constitute compelling proof that a particular property always holds. Methods based on logical inference uses formal version of mathematical reasoning to reason about the model in hand and is usually driven by the users understanding of the intricacies involved in the model.

In this thesis we adopted the technique based on logical inference to specify and verify the DNSSEC protocol. In this logical inference technique we can make use of different logics like BAN logics to derive the goals of the given security protocol. BAN logic [9] is an ancestor of other logics like AT [1], GNY [13] and SVO [21]. In this thesis we propose to use the SVO logic [21] for formal reasoning and verification of the DNSSEC protocol. SVO logic was designed to capture the features and extensions of four variants of logics, namely `BAN, GNY, AT, VO`, in a single unified framework. *SVO logic* was designed in such a way that, it is simple to use and more expressive than any of the logics from which it is derived; more details are discussed. Logics like SVO are particularly suitable for protocol verification because they are comparatively simple and effective.

In order to reason about an authentication protocol like DNSSEC using *SVO* logic we need to follow five step process. In the first step, we write initial conditions of the protocol, which includes the beliefs held by the principals and the possession of keys or any other prior assumptions the principals might be holding. In the second step we write the messages that are exchanged between principals which include the received messages, that is, the messages are that not lost in transit. In the third step we assert that the messages that are received by the principals are comprehended (eg. decryption of received message by using a key that is known already). In the fourth step we write the inferred messages by principals with the received messages and the knowledge that principals initially have. In the fifth step we use axioms and premises to derive the goals of the protocol. If we succeed in deriving the specified goals of the protocol then we can say that the protocol is working correctly according to the specification, else we can have suspicion about the protocol's correctness.

One has to be very careful while writing the initial conditions i.e., formalization of the protocol messages. Mistakes committed while formalizing the protocol messages and initial conditions render the verification process worthless. Detailed solution of formal analysis of DNSSEC protocol is discussed. As the complexity of a security protocol grows, it becomes a very difficult and tedious task to verify the correctness manually. To cope up with the complexity and ensuring of security functions of a given protocol, automatic validation tools [6, 11] are necessary. Automated validation tools for security protocols also play vital role in analyzing the working procedure of a protocol, specifically in parallel sessions and finding subtle errors.

In this thesis we propose to use the AVSPA tool to verify the DNSSEC security protocol. Automated Validation of Internet Security Protocols and Applications

(AVISPA) [2, 22] is an automatic validation tool used to verify the correctness of security properties of Internet security protocols. AVISPA uses a High Level Protocol Specification Language (HLPSL) [10, 4] in which the protocol to be verified is specified. HLPSL is a very expressive and intuitive language used to formalise any authentication protocols so that the AVISPA tool can be used for verifying the correctness. More details are discussed. HLPSL is basically a role based language, which means that each participant in a security protocol is represented as one role in HLPSL specification. In general, HLPSL specification of a protocol consists of one or more basic role definitions, followed by the definition of composite role. In a composite role, one or more basic roles are combined together so that they can be executed together, usually, in parallel. After composite role, an environment role is defined by instantiating the composition role to create one or more sessions. Finally, we specify the security parameters as goals that we want to verify by the back end tools. AVISPA currently supports four backend tools to validate the protocol. They are On-the-fly Model-Checker (OFMC) [5], SAT-based Model-Checker (SATMC) [3], Constraint-Logic-based Attack Searcher (CL-AtSe) [23] and Tree Automata based on Automatic Approximations for the Analysis of Security Protocols(TA4SP) [7].

The AVISPA tool accepts input specification in HLPSL and translates HLPSL specifications into Intermediate Format (IF). IF is an intermediate and independent form of a given specification, which can be later used by back-end tools for analysis. The IF format of a protocol is executed over finite number of times or entirely if no loop exist. Finally, the execution may end-up with either by an identification of an attack on a given protocol or the protocol is proved as safe over a given number of sessions.We have represented the DNSSEC protocol in HLPSL and ensured the correctness of both the security parameters, i.e., data integrity and data origin authentication. The specification and validation details of DNSSEC protocol are explained in the later Chapters.

We are aware that the authentication, specifically in networking, is temporal. It means that the session keys, One time passwords (OTPs) etc. which are used for authentication are valid only for a limited period of time. In order to utilise the authentication information more effectively and improve the performance of DNSSEC protocol, in this thesis we have proposed DNSSEC with temporal BDI. In a BDI system [12, 20], Beliefs are basically facts about the environment in which a system exists, Desires are the goals the system want to achieve and Intentions are the goals which are currently under consideration. Preparation of a plan to achieve the goal is the important task in BDI system. Method of plan generation depends on the problem to be solved. In general a plan is a sequence of actions to be executed for achieving the goal. In order to find plans for the problems that deal with time, temporal planning technique is used. In real time problems, planning with time is very important as it means that performing a poor action within a specified time is preferable to performing a planned action which is too late. The planner has to generate a plan that is optimal and satisfies user constraints.

Our aim in this thesis is to extend the basic BDI architecture [8] and the classical planning life cycle to incorporate *temporal planning*. After proposing and analysing temporal planing in BDI system, we have utilised this technique for DNSSEC protocol to improve the performance. DNSSEC protocol uses temporal planning method to establish *chain of trust*. It is obvious and apparent that many requests and responses are generated and transmitted over a network during construction of chain of trust. In order to avoid unnecessary construction of chain of trusts, authentication data obtained during construction of chain of trust are saved as beliefs in BDI system for a valid time period. Authentication data saved in BDI system can be used by the client of DNSSEC protocol to answer the relevant queries without reconstruction of the chain of trust and thereby improve the performance of DNSSEC protocol as well as reducing Internet traffic. This will also reduce the response time of the DNS query. The details regarding the usage of BDI-based temporal planning to improve the performance of the DNSSEC protocol are discussed in Chapter 5.

4 Thesis Contributions

As mentioned in the previous section, In this thesis we provide reasoning about the security mechanisms involved in the DNSSEC protocol. DNSSEC protocol uses *chain of trust* technique to ensure *data integrity* and *origin authentication* of received data. In order to ensure that the process of chain of trust is formally correct, SVO logic based specification and verification method is proposed. *The results related to this work has been published in the form of a conference paper titled* ***Reasoning about DNSSEC*** *in the Fifth Multidisciplinary International Conference on Artificial Intelligence (MIWAI), 2011.*

In order to validate the DNSSEC protocol at a low level of abstraction in-terms of parallel sessions and other constraints we demonstrate how the AVISPA (Automatic Verification of Internet Security Protocols and Applications) tool can be used. AVISPA tool is an automatic verification tool for internet security protocols. In the process of validation, first we have represented the DNSSEC protocol using HLPSL (High Level Protocol Specification Language) language. AVISPA translates the HLPSL specification into IF (Intermediate Form) format. The IF form is then analyzed by using various model checkers like SATMC, OFMC, CL-Atse, TA4SL to ensure security parameters. *The results related to this work has been accepted at the International Conference on Computing Analytics and Networking (ICCAN) - 2017 with the title* ***Automated Validation of DNSSEC*** *.*

The standard BDI architecture is extended to incorporate temporal planning. Generating a plan for a given problem is one of the important procedures in BDI system. Certain problems need only the plan, whereas other problems need a plan which can complete the task in a given time. When time plays an important role we need to go for techniques that can account for temporal planning in order to generate a plan. We propose a Temporal planning framework for the standard BDI architecture. *The results of this work is published as* ***Incorporating Temporal Planning Within a BDI Architecture*** *at the 5th Indian International Conference on Artificial Intelligence (IICAI 2011).*

The proposed Temporal planning BDI model is extended to apply in the case of the DNSSEC protocol so as to improve the performance of the authentication mechanism in DNSSEC. Temporal planning technique is further used to establish the chain of trust in DNSSEC. BDI-based architecture is used to save authentication information for a valid amount of time so that when a request comes the information related to the request is first checked with the saved authentication information. *The results related to this work has been accepted at the 14th International Conference on Dis-*

tributed Computing and Internet Technology (ICDCIT-2018 with the tile ***`BDI-based Performance Enhancement in DNSSEC`****).*

5 Thesis Organization

Apart from the first chapter which has an introductory tone this thesis has five more Chapters. Chapter 2 outlines the foundational concepts as well as related work on which this thesis is built. We start Chapter 2 with an introduction to the DNS Infrastructure wherein we talk about concepts related to domain name space, resource records, name servers and resolvers. We also give a brief description of the DNS workflow and Various kinds of attacks that can happen on DNS and thereafter we give a brief introduction to the DNSSEC protocol. The importance of formal methods is discussed in detail and the use of several variants of Modal logics used for specification and verification of security protocols is also explained. Chapter 2 ends with the exploration of various validation tools available in the literature that can be used for validating security protocols. In Chapter 3 we proposed a formal framework based on Modal logic called SVO logic that can be used to specify and verify the DNSSEC protocol. We proposed an extension of DNSSEC wherein two new axioms were added to the original SVO logic and by making use of the inference rules as well as axioms of SVO logic we were able to successfully derive proof for Data origin authentication and Data integrity of DNSSEC protocol. To the best of our knowledge we were the first ones to propose a logic based framework for analysing the DNSSEC protocol.

Contributions related to the development of validation tools for analysing the logical specifications of DNSSEC as developed in Chapter 3 was carried out in Chapter 4. We demonstrated how the DNSSEC protocol specification in Alice - Bob (A - B) notation can be verified using the AVISPA tool. Chapter 5 is a lengthy chapter in which we have two parts and in which we make two important contributions. The first part tries to address the analysis and verification of DNSSEC from a Multi-Agent perspective. In this part we extend a particular agent architecture called the BDI so as to incorporate temporal planning with explicit time-limits. To this end we develop a temporal BDI architecture and demonstrate with the help of examples how temporal plan can be generated within a BDI architecture. In the second part we use this temporal BDI model for performance enhancement of the DNSSEC protocol. We conclude the thesis with Chapter 6.

6 List of Publications

1. Kollapalli Ramesh Babu, Vineet Padmanabhan and Wilson Naik Bhukya, *Reasoning about DNSSEC*, Fifth Multidisciplinary International Conference on Artificial Intelligence (MIWAI 2011), pages 75 - 86, LNAI, Springer.

2. Vineet Padmanabhan, Abdul Sattar, Guido Governatori, and Kollapalli Ramesh Babu, *Incorporating temporal planning within a BDI architecture*, 5th Indian International Conference on Artificial Intelligence (IICAI 2011) pages 1618 - 1636.

3. Kollapalli Ramesh Babu and Vineet Padmanabhan, *Automated validation of DNSSEC*, Accepted in the International Conference on Computing Analytics and Networking (ICCAN 2017), 15-16 Decmber-2017, Odhisa.

4. Kollapalli Ramesh Babu and Vineet Padmanabhan, *BDI based performance enhancement in DNSSEC*, Accepted in the 14th International conference on Distributed Computing and Internet technology (ICDCIT 2018), 11-13 January-2018, Odhisa.

References

[1] Martin 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, 1991.

[2] Alessandro Armando, David Basin, Yohan Boichut, Yannick Chevalier, Luca Compagna, Jorge Cuéllar, P Hankes Drielsma, Pierre-Cyrille Héam, Olga Kouchnarenko, Jacopo Mantovani, et al. The avispa tool for the automated validation of internet security protocols and applications. In *International conference on computer aided verification*, pages 281–285. Springer, 2005.

[3] Alessandro Armando and Luca Compagna. Satmc: a sat-based model checker for security protocols. In *JELIA*, volume 3229, pages 730–733. Springer, 2004.

[4] Team AVISPA. Hlpsl tutorial, 2014.

[5] David Basin, Sebastian Mödersheim, and Luca Vigano. Ofmc: A symbolic model checker for security protocols. *International Journal of Information Security*, 4(3):181–208, 2005.

[6] Bruno Blanchet and Andreas Podelski. Verification of cryptographic protocols: Tagging enforces termination. In *FOSSACS'03/ETAPS'03*, pages 136–152. Springer, 2003.

[7] Yohan Boichut, Pierre-Cyrille Héam, and Olga Kouchnarenko. *Automatic verification of security protocols using approximations.* PhD thesis, INRIA, 2005.

[8] Rafael H Bordini, Ana LC Bazzan, Rafael de O Jannone, Daniel M Basso, Rosa M Vicari, and Victor R Lesser. Agentspeak (xl): Efficient intention selection in bdi

agents via decision-theoretic task scheduling. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 3*, pages 1294–1302. ACM, 2002.

[9] Michael Burrows, Martin Abadi, and Roger M Needham. A logic of authentication. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, volume 426, pages 233–271. The Royal Society, 1989.

[10] Yannick Chevalier, Luca Compagna, Jorge Cuellar, Paul Hankes Drielsma, Jacopo Mantovani, Sebastian Mödersheim, and Laurent Vigneron. A high level protocol specification language for industrial security-sensitive protocols. In *Workshop on Specification and Automated Processing of Security Requirements-SAPS'2004*, pages 13–p. Austrian Computer Society, 2004.

[11] Cas J. F. Cremers. *The Scyther Tool: Verification, Falsification, and Analysis of Security Protocols*, pages 414–418. Springer Berlin Heidelberg, 2008.

[12] M dInverno and M Luck. A formal specification of agentspeak (l). *Logic and Computation*, 8(3), 1998.

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

[14] Gavin Lowe. An attack on the needham-schroeder public-key authentication protocol. *Inf. Process. Lett.*, 56(3):131–133, 1995.

[15] C. Meadows. A procedure for verifying security against type confusion attacks. In *16th IEEE Computer Security Foundations Workshop, 2003. Proceedings.*, pages 62–72. IEEE, 2003.

[16] P. Mockapetris and K. J. Dunlap. Development of the domain name system. *SIGCOMM Comput. Commun. Rev.*, 18(4):123–133, 1988.

[17] Paul V Mockapetris. Domain names: Implementation specification. RFC 1035, Internet Engineering Task Force, 1983.

[18] Paul V Mockapetris. Domain names-concepts and facilities. RFC 1034, Internet Engineering Task Force, 1987.

[19] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21(12):993–999, 1978.

[20] Anand S Rao. Agentspeak (l): Bdi agents speak out in a logical computable language. In *European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, pages 42–55. Springer, 1996.

[21] Paul F Syverson and Paul C Van Oorschot. On unifying some cryptographic protocol logics. In *Research in Security and Privacy, 1994. Proceedings., 1994 IEEE Computer Society Symposium on*, pages 14–28. IEEE, 1994.

[22] TA Team et al. Avispa v1. 1 user manual. *Information Society Technologies Programme (June 2006), http://avispa-project. org*, 2006.

[23] Mathieu Turuani. The cl-atse protocol analyser. In *International Conference on Rewriting Techniques and Applications*, pages 277–286. Springer, 2006.