Cryptanalysis of Elliptic Curve Discrete Logarithm Problem

A thesis submitted in partial fulfillment
of the requirements for the award of
Doctor of Philosophy in Computer Science

by

N.Anil Kumar
04MCPC01

School of Computer & Information Sciences
University of Hyderabad
Hyderabad

2015

CERTIFICATE

This is to certify that the thesis entitled **'Cryptanalysis of Elliptic Curve Discrete Logarithm Problem'** being submitted by **N. Anil Kumar** bearing Reg. No: **04MCPC01** for the award of the degree of Doctor of Philosophy in Computer Science to the University of Hyderabad, is a record of bonafide and original work carried out by him under my supervision.

The matter embodied in this thesis has not been submitted to any other University or Institute for award of any degree or diploma.

Prof. Chakravarthy Bhagvati
Supervisor

Dean
School of Computer and Information Sciences

DECLARATION

I (N. Anil Kumar) hereby declare that the thesis entitled “Cryptanalysis of Elliptic Curve Discrete Logarithm Problem” being submitted by me is a bonafide research work carried out by me. 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:
Signature of the Student:
Reg. No:

Contents

List of Algorithms

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Prof. Chakravarthy Bhagvati for his guidance, support throughout the course of the thesis. I would like to thank Prof Arun Agarwal, Dean, School of Computer & Information Sciences for the support, help and the encouragement he has provided to for submitting the thesis. I would like to thank the Doctoral Review Committee(DRC) members Dr. S. Durga Bhavani and Prof H. Mohanty for their suggestions.

I would like to thank Prof. Rajat Tandon for teaching me Algebra and Number theory. Because of the personal care taken by Prof. Rajat Tandon, I was able to understand the basics of Algebra, Number theory and cryptography. I would like to thank Dr. Eerke. A. Boiten, Director, Cyber Security Research Center, University of Kent, United Kingdom for his comments and suggestions which helped in shaping this thesis. I would like to thank Dr. Vishal Saraswat, C R Rao AIMSCS, for giving suggestions.

I would like to thank IISc Mathematics Initiative (IMI) for organising workshop on Number Theory and Cryptography, Microsoft Research Summer School which helped me a lot in understanding cryptography.

I would like to thank my friends Subrath Kumar Dash, Neelakanta for their support, help and suggestions. I would like to thank Dr. Y V Subba Rao for his support. I would like to thank the extended committee members Prof. H. Mohanty, Prof. C. Raghavendra Rao, Prof. S.K. Udgata and Dr. S. Durga Bhavani for their suggestions and help.

I would like to thank Prof. S. B. Rao, Principal Investigator DST-CMS, C R Rao AIMSCS for his support and encouragement. I would like to thank Dr. A Nagesh, Professor & HoD, Department of Computer Science & Engineering, Mahatma Gandhi Institute of Technology, Hyderabad for his support and encouragement.

I would like to thank ISRO RESPOND project, CISR-SRF, for their financial support. I would like to specially thank DST-CMS project Lr.No.SR/S4/MS:516/07, Dt.21-04-2008 for their financial support.

ABSTRACT

Elliptic curves offer all the features of the conventional cryptography like digital signatures, data encryption / decryption, key exchange, identification, etc. at a reduced key size. Many cryptosystems are built on the assumption that Elliptic Curve Discrete logarithm problem (ECDLP) is hard from computational point of view. Standardized elliptic curve based signature schemes are described in ANSI X9.62, IEEE 1363-2000, etc.

We propose a Two key digital signature scheme using elliptic curves and compare our scheme with the standard scheme. We have developed an algorithm for attacking the proposed Two key signature scheme. We have suggested an application of the proposed two key signature scheme in Digital Certificates, Counter signing of documents.

Side channel attacks are used to reveal information about the key if proper countermeasures are not used. We study the discrete logarithm problem assuming that partial information about the key is known. K.Gopalakrishnan et al. [16] provided algorithms to solve the discrete logarithm problem for generic groups which are considerably better than square-root attack on the whole key when one contiguous bits of the key is revealed. We developed algorithms which will work when more than one portion of key i.e. some parts of the key are revealed of the key are revealed with time complexity of order square root $2^{(\text{unknown bits})}$. We have analysed the proposed algorithm both theoretically and empirically. We have developed a parallel version of the proposed algorithm to solve discrete logarithm from partial knowledge of the key.

Chapter 1

Introduction

Discrete Logarithm Problem (DLP) forms the basis for many cryptographic systems today. The problem involves finding the exponent (α) given the result (β) and the base (g). In other words, g and β are given in the equation $\beta = g^{\alpha}$ and it is required to find α. This problem is a hard computational problem on many mathematical structures such as finite fields, Galois fields, etc. and offers an attractive alternative to the problem of factoring large numbers that is used in the RSA public key cryptosystem. As a result, many algorithms have been defined in literature for solving the DLP but only Index-Calculus methods offer sub-exponential time complexities for DLP on finite fields.

Elliptic curves, whose equations are of the form $y^2 = x^3 + ax + b$ possess many interesting mathematical properties that are exploited by the cryptographic community to develop cryptosystems that are more secure than those relying on finite or Galois fields. Points on an elliptic curve form group structures that are rich enough to allow DLP to remain as a hard computational problem while not having certain other properties that allow powerful attacks to mounted on DLP. For example, Index-Calculus methods cannot directly defined on elliptic curves for solving DLP and today there are no known sub-exponential time algorithms for solving DLP on many classes of elliptic curve groups.

In this thesis, the focus is on the DLP on elliptic curves, i.e., the Elliptic Curve Discrete Logarithm Problem (ECDLP). The work studies the use of ECDLP in developing cryptosystems in general and on *signature schemes* in particular. A two key signature scheme is proposed as an extension to an existing single key scheme. Several applications, such as countersigning of documents, provide a natural context for using the multi-key signature scheme.

A second part of the work in this thesis studies algorithms for solving ECDLP. In particular, the work generalises and extends an algorithm that uses partial knowledge of the

secret key. Such knowledge may be available through implementation details or side channel analysis or both. The work extends the existing algorithm to the case when different portions of the key are known. In addition to these two major studies, there are several smaller contributions which are discussed in Chapter 3 and Chapter 5.

1.1 Problems addressed

Problems addressed in the thesis, from cryptography point of view we have designed a secure two key signature scheme and from cryptanalysis point of view we have improved and extended the Algorithm to Solving Discrete Logarithms from Partial Knowledge of the Key.

We want to design a new problem similar to Discrete Log problem (DLP) on which cryptanalytic techniques like Pohlig-Hellman does not work and which has two keys. We want to show its usefulness by designing a signature scheme based on the new problem and show some applications.

Gopalakrishna et al. [16] has proposed an algorithm that improves on the exhaustive search when one contiguous bits of the key is revealed. We want to design an algorithm which improves over Gopalakrishna et al. and also works when more than one part (i.e random bits) of the key are revealed.

The scope of the thesis is limited to design and analysis of mathematical security aspects and computational security aspects of the proposed cryptosystem and proposed algorithm.

1.2 Contributions

The contributions of the thesis can be summarised as

1. We have analysed the existing Elliptic Curve Digital Signature Scheme(ECDSA) and proposed a new signature scheme, Two key signature scheme
2. We have observed that the newly proposed, Two key signature scheme has application in Public Key Infrastructure (PKI), counter signing applications, etc. We have analysed the X.509 digital certificate format and explained in detail how two key signature scheme can be used in digital certificates. We have slightly modified the signing algorithm of two key digital signature scheme so that it can be used in counter signing.

3. We have developed an algorithm for attacking the proposed Two key signature scheme and named it the modified Shanks algorithm.

4. We have analysed in detail the algorithms for attacking ECDLP and in particular the partial key attack by Gopalakrishnan et al. [16] and found an error in the algorithm which has been corrected.

5. We have improved and extend the Gopalakrishnan et al. [16] algorithms. Gopalakrishna et al. [16] has proposed an algorithm that works when one portion of the key is revealed. We designed an algorithm which improves over Gopalakrishna et al. and also works when more than one portion (i.e random bits) of the keys are revealed.

6. We have developed a parallel version of the proposed algorithm to solve discrete logarithm from partial knowledge of the key.

7. We have developed a library using C and GMP [14] for implementing the proposed algorithms. We have implemented the proposed algorithms and published algorithms and has given a comparative analysis.

1.3 Organisation of Thesis

The thesis is organised as follows.

Chapter 1: Introduction This chapter introduces the significance of ECDLP. The problems addressed in the thesis and the contributions of the thesis.

Chapter 2: Elliptic Curve Discrete Logarithm Problem This chapter explains the group structure of the elliptic curves, introduces the concept of point at infinity. This chapter also introduces Elliptic Curve Discrete Logarithm Problem (ECDLP) and various schemes based on ECDLP such as Diffie-Hellman key exchange protocol, Elliptic Curve Integrated Encryption Scheme, ECMQV key agreement scheme, etc. This chapter also describes the attacks such as Baby step - Giant step algorithm, Pohlig-Hellman algorithm, Pollard ρ method, Pollard lambda method, MOV attack etc., in detail. In addition, a brief introduction to Sage and GMP libraries is presented.

Chapter 3: Two key Elliptic Curve Digital Siganture algorithm This chapter describes about ECDSA published in ANSCI X9.62 and our proposed Two key Elliptic curve Digital Siganture algorithm. Some application of the proposed scheme are also discussed. This chapter also describes an algorithm for attacking the proposed Two key signature scheme.

Chapter 4: Solving Discrete Logarithm from partial knowledge of the key This chapter introduces the notation used by Gopalakrishnan et. al [16] and their approach in solving the DLP problem from partial knowledge.

Chapter 5: Efficient Algorithm to Solving Discrete Logarithms from Partial Knowledge of the Key This chapter describes about our proposed solution to the partial knowledge of the key problem. Gopalakrishna et al. [16] has proposed an algorithm that improves on the exhaustive search when one contiguous bits of the key is revealed. The error in the Gopalakrishna et al. algorithm is corrected and is explained in this chapter. We designed an algorithm which improves over Gopalakrishna et al. and also works when more than one part (i.e random bits) of the keys are revealed which is explained in detail in this chapter. We also developed a parallel version of this algorithm.

Chapter 6: Conclusion This chapter summarises the work done and gives the concluding remarks of the thesis.

Chapter 2

Elliptic Curve Discrete Logarithm Problem

The main topic of the present study is Cryptanalysis of the Elliptic Curve Discrete Logarithm. This chapter introduces arithmetic properties of Elliptic curves such as group structure, point at infinity, etc. This chapter also introduces Elliptic Curve Discrete Logarithm Problem (ECDLP) and various schemes based on ECDLP such as Diffie-Hellman key exchange protocol, Elliptic Curve Integrated Encryption Scheme, ECMQV key agreement scheme, etc. This chapter also describes the attacks such as Baby step - Giant step algorithm, Pohlig-Hellman algorithm, Pollard ρ method, Pollard lambda method, MOV attack etc., in detail. In addition, a brief introduction to Sage and GMP libraries is presented as these are used for implementing the algorithms presented in Chapter 3, 4 and 5.

2.1 Arithmetic of Elliptic Curves

This section closely follows [11, 30]

Elliptic curve E is the graph of an equation of the form

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

where $a_1, \ldots, a_6$ are constants belonging to some field K, called a invariants. This equation is called the generalized Weierstrass equation. If the characteristic of the field is not 2, then

we can divide by 2 and complete the square

$$\left(y + \frac{a_1 x}{2} + \frac{a_3}{2}\right)^2 = x^3 + \left(a_2 + \frac{a_1^2}{4}\right) x^2 + \left(a_4 + \frac{a_1 a_3}{2}\right) x + \left(\frac{a_3^2}{4} + a_6\right)$$

which can be written as

$$y_1^2 = x^3 + a_2' x^2 + a_4' x + a_6'$$

with $y_1 = y + a_1 x/2 + a_3/2$ and with some constants a_2', a_4', a_6'. If the characteristic is also not 3, then we can let $x_1 = x + a_2'/3$ and obtain

$$y_1^2 = x_1^3 + a x_1 + b$$

for some constants a and b. So the simplified Weierstrass equation for elliptic curve is

$$y^2 = x^3 + ax + b \tag{2.1}$$

where a, b, x and y belong to some field with $4a^3 + 27b^2 \neq 0$.

2.1.1 Elliptic Curve group

Let P and Q be two points on the elliptic curve. Let the line joining the two points intersect the curve at a third point say R. The addition of points is defined as

$$P + Q + R = \text{identity}$$

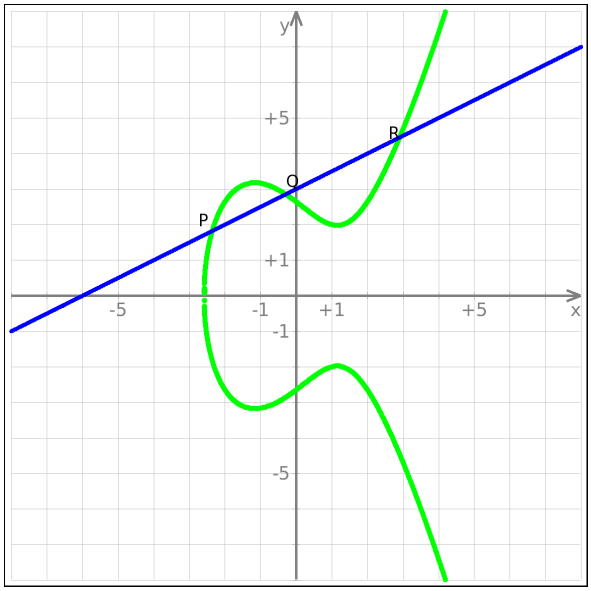
Identity is defined as a point at infinity denoted by $\mathcal{O}$. A line is said to pass through point at infinity when it is vertical. It may easily be seen that the elliptic curve forms a group under addition.

Scalar multiplication is defined a repeated addition. Let n be integer.

$$nP = P + P + \cdots \text{n times}$$

The following holds when P, Q and R are points on the elliptic curve.

- $P + Q$ will be point on the curve (Closure)
- $P + Q = Q + P$ (Commutativity)



- $(P + Q) + R = P + (Q + R)$ (Associativity)
- $P + \mathcal{O} = \mathcal{O} + P = P$ (Existence of an identity element)
- There exists $(-P)$ such that $-P + P = P + (-P) = \mathcal{O}$ (Existence of inverse)

The points on the elliptic curve form an abelian group.

Let $P(x_1, y_1) + Q(x_2, y_2) = R(x_3, y_3)$ then

$$\begin{aligned} x_3 &= m^2 - x_1 - x_2 \\ y_3 &= m(x_1 - x_3) - y_1 \end{aligned}$$

where

$$\begin{aligned} m &= \frac{y_2 - y_1}{x_2 - x_1} \text{ for } P \neq Q \\ &= \frac{3x_1^2 + a}{2y_1} \text{ for } P = Q \end{aligned}$$

The elliptic curves which have been defined over real numbers can also be defined over finite fields. Most of the definitions over real numbers can be carried over to finite fields. [20, 30] give a detailed description on the arithmetic of elliptic curves.

Let $q = p^k$ where p is prime. $\mathbb{F}_q$ is the Galois field of order q. Let E be the elliptic curve.

The set of points on the elliptic curve is denoted by $E(F_q)$.

$$E(F_q) = \{\mathcal{O}\} \cup \{(x, y) \in F_q \times F_q \mid y^2 = x^3 + ax + b\}$$

2.1.2 Weil Pairing

The Weil pairing on an elliptic curve is a major tool in the study of elliptic curves.

Let E be an elliptic curve over a field K and let n be an integer not divisible by the characteristic of K. Let $\mu_n = \{x \in K | x^n = 1\}$. The pairing

$$e_m : E[n] \times E[n] \to \mu_n$$

is called Weil pairing.

2.1.3 Millers algorithm

Millers algorithm can be used for computing the Weil pairing. Fix an integer m coprime to p and let $U, V \in E[m]$. Let C and D be divisors such that $C \sim \langle U \rangle - \langle \mathcal{O} \rangle$ and $D \sim \langle V \rangle - \langle \mathcal{O} \rangle$, C and D have disjoint support. Since U and V are m- torsion points, it easily follows that mA and mB are principal divisors. So there are f_C and $f_D \in k(E)$ such that
$div(f_C) = mC$ and $div(f_D) = mD$.
Weil Pairing:

$$e_m(U, V) = \frac{f_C(D)}{f_D(C)}$$

We want to compute the weil pairing of the m - torsion points U, V $\in E[m] \subseteq E(k)$.
Choose points $U', V' \in E(k)$ such that $U', V', U' + U, V' + V$ are all different.
Set C= $\langle U' + U \rangle - \langle U' \rangle$ and D= $\langle V' + V \rangle - \langle V' \rangle$
div(f_C) = $mA = m\langle U' + U \rangle - m\langle U' \rangle$ and
div(f_D) = $mB = m\langle V' + V \rangle - m\langle V' \rangle$

$$e_m(U, V) = \frac{f_C(D)}{f_D(C)} = \frac{f_C(\langle V + V' \rangle - \langle V' \rangle)}{f_D(\langle U + U' \rangle - \langle U' \rangle)} = \frac{f_C(V + V') f_D(U')}{f_C(V') f_D(U + U')}$$

2.2 Discrete Logarithm

An ordinary logarithm $\log_g(\beta)$ is a solution of the equation $g^\alpha = \beta$ over the real or complex numbers. Similarly, if g and β are elements of a finite cyclic group G then a solution α of the equation $g^\alpha = \beta$ is called a *Discrete Logarithm Problem* (DLP) in group G.

Definition 2.1 *Let G be a finite cyclic group with n elements. Let g be the generator of G, then every element, β of G can be written in the form $g^\alpha = \beta$ for a multiplicative group and $g \times \alpha = \beta$ for an additive group for some α. Thus we can define a function*

$$log_g : G \to \mathbb{Z}_n$$

where $\mathbb{Z}_n$ is the ring of integers modulo n.

This function is a group isomorphism, called the discrete logarithm to base g [24].

Let E be the elliptic curve defined over F_q. Let g, β be points on the elliptic curve E such that $\alpha\ g = \beta$, finding α given g and β is called *Elliptic Curve Discrete Logarithm Problem* (ECDLP). No algorithms are known to solve the Discrete Log Problem and Elliptic Curve Discrete Log Problem in polynomial time on the size of the underlying field [1].

2.3 Schemes based on Elliptic Curve Discrete Logarithm Problem

Finding β given α and g is easy but calculating α given g and β is difficult. Thus, it is a potential candidate for one-way functions and numerous cryptosystems have been designed on the difficulty of solving Elliptic Curve Discrete Log Problem. Some cryptosystems are

1. Diffie-Hellman key exchange protocol
2. ElGamal cryptosystem
3. Elliptic Integrated Curve Encryption scheme
4. ECMQV key agreement scheme
5. ECQV implicit certificate scheme
6. ElGamal signature scheme

7. Schnorr signature algorithm and others

Diffie-Hellman key exchange protocol

Diffie-Hellman key exchange (D-H) [10] is an earliest protocol to exchange the keys between two parties. The Diffie-Hellman key exchange method allows two parties that have no prior knowledge of each other, but share some public parameters to jointly establish a shared secret key over an insecure communications channel. This key can then be used to encrypt subsequent communications using a symmetric key cipher.

The scheme was first published by Whitfield Diffie and Martin Hellman in 1976, although it had been separately invented a few years earlier within Government Communications Headquarters (GCHQ), the British signals intelligence agency, by James H. Ellis, Clifford Cocks and Malcolm J. Williamson but was kept classified [2, 31].

A cryptographic key is called **ephemeral** if it is generated for each execution of a key establishment process.

Forward secrecy is obtained by generating new key for each session, that is generating an ephemeral key to be used for all messages of a conversation. If the adversary got access to the device or system he can only get the key for the session, but not the keys for earlier session.

Perfect forward secrecy If we generate new key for each message of the conversation even if the attacker got access to the system or device he can retrieve only one message but not the previous messages in the same session. So this is called Perfect forward secrecy.

They will be using this integer as a secret key (or used to derive secret key) for further communication using symmetric encryption schemes like AES, Triple DES, RC4, Trivium, etc. The Diffie-Hellman key exchange protocol is used to establish a common key between Alice and Bob who are communicating through a public channel, may be eavesdropped by a third party.

Let g be a generator of the group $\mathbb{Z}_p^*$. Both the communicating parties agree upon the generator and the group. Alice selects a random integer say a in $\mathbb{Z}_p^*$ (integers like 1, -1 are avoided). She computes $A = g^a \mod p$ and sends A to Bob. Bob selects a random integer b and computes $B = g^b \mod p$. He sends B to Alice. Now Alice computes $B^a \mod p$ which is $g^{ab} \mod p$. Similarly Bob computes $A^b \mod p$ which is $g^{ab} \mod p$. Now they

have a shared secret which can be used for futher communication.

Computational Diffie-Hellman(CDH) assumption states that in a cyclic group like $\mathbb{Z}_p^*$ of generator g given (g, g^a, g^b) in group $\mathbb{Z}_p^*$ for randomly chosen generator g and random a, b. it is computationally intractable to compute the value of g^{ab} in group $\mathbb{Z}_p^*$.

Under Computational Diffie-Hellman assumption the attacker who knows (g, g^a, g^b) cannot computer g^{ab}.

Elliptic Curve Diffie-Hellman

We can use Diffie-Hellman protocol on Elliptic Curve groups as well.

Let $E(F_q)$ be the Ellipitc Curve group. Let g be the generator of the group. Both the communicating parties agree on the elliptic curve group and the generator. Alice selects a random integer say a and computes $A = a \times g$ and sends A to Bob. Bob selects a random integer b and computes $B = B \times g$. He sends B to Alice. Now Alice computes $a \times B$ which is $a \times b \times g$. Similarly Bob computes $b \times A$ which is $a \times b \times g$. Now both the parties have a shared point between them. They can use x-coordinate or y-coordinate or a function of both the coordinate to derive a secret key.

Under Computational Diffie-Hellman assumption the attacker who knows $(g, a \times g, b \times g)$ cannot computer $ab \times g$.

Elliptic Curve Integrated Encryption scheme

Integrated Encryption Scheme (IES) [28] is a encryption scheme which is secure against an adversary who is allowed to use chosen-plaintext and chosen-ciphertext attacks. The security of the scheme is based on the DiffieHellman problem.

ECMQV key agreement scheme

MQV (Menezes - Qu - Vanstone) [18] is an protocol for key agreement based on the Diffie-Hellman scheme. This scheme provides protection against an active attacker. The protocol can be used on ellitpic curve groups also.

In this section we have described about the various schemes based on the ECDLP. We

will be describing about Elliptic Curve Digital Signature Algorithm (ECDSA), a scheme which is based on ECDLP, in the next Chapter along with the proposed Two Key Elliptic Curve Digital Signature Algorithm (Two Key ECDSA). We will do the security analysis and propose application for Two Key ECDSA.

2.4 Algorithms for solving discrete log problem

1. Shanks' Algorithm
2. Pohlig-Hellman Algorithm
3. Pollard ρ method
4. Pollard λ method
5. Index Calculus method
6. MOV Algorithm

All these attacks, except Index Calculus, work on both DLP and ECDLP. The advantage with the index calculus method is much faster (subexponential) than the remaning algortihms (exponential). MOV is an algorithm, which is based on Weil pairing which converts elliptic curve discrete logarithm problem to discrete logarithm over finite fields. So we can use Index Calculus to solve it.

Baby-Step, Giant-Step algorithm

D.Shanks developed this method, also called Shanks' algorithm. The algorithm is based on a space-time trade-off. Given a cyclic group G and a generator g of order N, the problem is to find an integer α such that

$$\alpha \cdot g = \beta$$

The baby-step giant-step algorithm is based on rewriting α as $\alpha = im + j$, with $m = \left\lceil \sqrt{N} \right\rceil$ and $0 \leq i < m$ and $0 \leq j < m$. Therefore we have

$$\beta - i \cdot m \cdot g = j \cdot g$$

The algorithm

1. Let $m \geq \sqrt{N}$ be an integer.
2. Generate $i \cdot g$ for $0 \leq i < m$ and store them in list 1.
3. Compute the points $\beta - jmg$ for j = $0, 1, \ldots m-1$ and compare with list 1.
4. If $ig = \beta - jmg$, we have $\beta = \alpha g$ with $k \equiv i + jm \bmod N$. else goto Step 3.

The point $i \times g$ is calculated by adding g to $(i-1)g$ this is called *baby* step. The point $\beta - jmg$ is computed by adding $-mg$ called the *gaint* step. Hence the name Baby Step, Gaint Step. The time complexity and space complexity of the algorithm is $O(\sqrt{N})$

2.5 Pollard ρ method

Pollard methods [15] for solving discrete logarithm problem can be classified as collision algorithms. Collision algorithms involve searching a space of keys, plaintexts or ciphertexts, or for public key cryptosystems, they may be aimed at solving the underlying hard mathematical problem. We would like to introduce few basics of probability for understanding collision algorithms.

Theorem 1 (Collision Theorem) *An urn contains N balls, of which n are black and $N-n$ are white. Bob randomly selects a ball from the urn, replaces it in the urn, randomly selects a second ball, replaces it, and so on. He does this until he has looked at a total of m balls.*

(a) The probability that Bob selects at least one black ball is

Pr(at least one black) = $1 - \left(1 - \frac{n}{M}\right)^m$

(b) A lower bound for the probability is Pr(at least one black) $\geq 1 - e^{-mn/N}$ □

We want to connect this theorem with the finding the match in two lists. Let the urn contains identical numbered white balls. If we pick a ball we note down the number and paint it black and replace it in the urn. So the probability of collision in the list is same as the probability of collision of black balls.

2.5.1 Abstract Formulation of Pollard's ρ method

The problem is to find $\alpha g = \beta$ given g and β. If we find a relation between g and β we can solve for α.

The idea of Pollard's ρ method is to generate numbers at random using a function which involves g and β. Let S be a finite set of points on the elliptic curve and let

$$f : S \to S$$

Let $f(g, \beta, x)$ be a random function. Let x be a point on the elliptic curve. $f^1 = f$, $f^2 = f(f(g, \beta, x))$ and so on. We generate f^1, $f^2, \ldots$ and keep checking the previous values. As the set S is finite we cannot get infinite point by repeatedly applying f. So there will be repetition. If a match is found, then we got a relationship involving g and β. So we can solve for α.

We have to store all the previous values of f^i and keep checking them. So that we find the collision. This requires the storage of $O(n)$. To keep the memory requirements low we can use Floyd's cycle detection algorithm. Checking i and $2i$ in a sequence is sufficient to find collision as proved in proposition 11.1 in [27]. So we generate f^1 and f^2 to check whether they are equal or not. Next we generate f^2 and f^4 to check whether they are equal or not. We keep on continuing the process till a match is found.

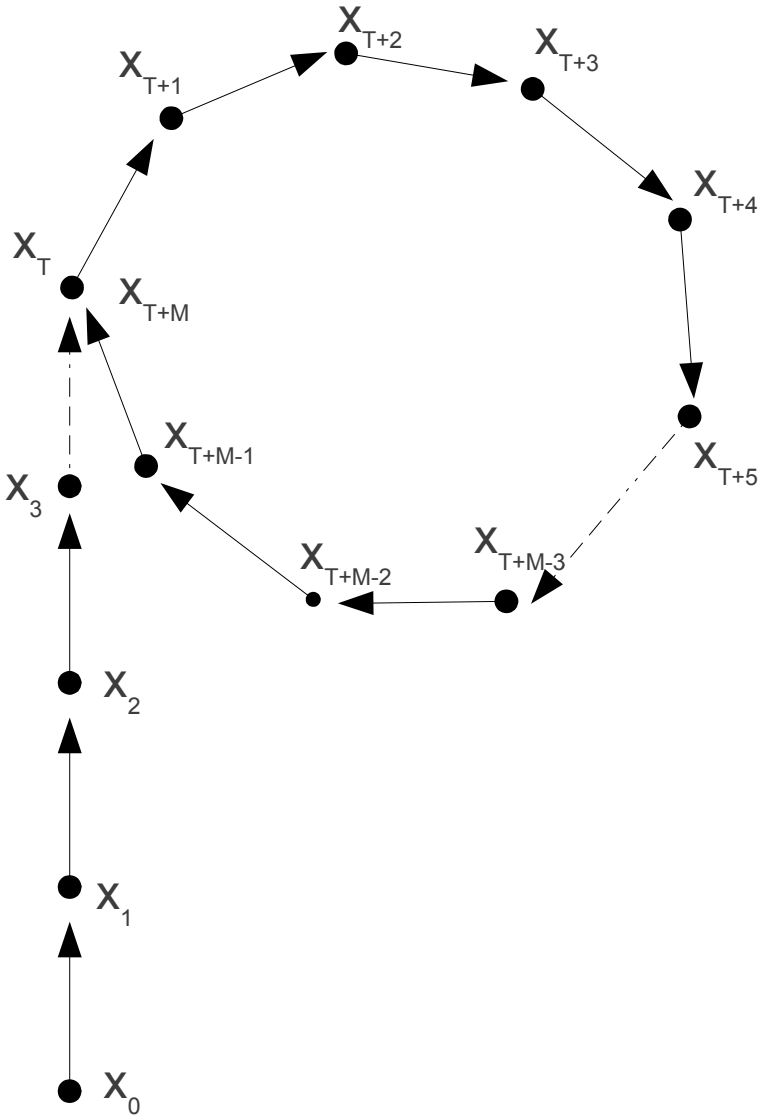


Figure 2.1: Pollard Rho

2.5.2 Algorithm for Pollard rho method

The equation which we are going to solve is $\alpha\, g = \beta$ where α is the key which is unknown g is the generator. g, β are called public parameters. We get the input of g and β, order of the curve from the user.

Input: g the generator of the elliptic curve.

β the public parameter.

order : the number of points on the curve

a, b, p : Parameter that define elliptic curve $y^2 = x^3 + ax + b$ defined over Z_p

Output: α

Data: struct state_index{

BigInt i, j;

EllipticCurvePoint point;

} rg, rbeta, r1, r2, t;

BigInt order;

EllipticCurve.setCurve(BigInt a,BigInt b, BigInt p);

setCurve is method of class EllipticCurve. /* We have to pass the parameters a, b and prime p of the elliptic curve $y^2 = x^3 + ax + b$ defined over Z_p to the method setCurve so that addition of points, scalar multiplication of points takes place on this curve. */

The decription of Pollard's Rho algorithm

rg.i=initlised to some random value ;

rg.j=initlised to some random value ;

rg.point=rg.i$\times g$+rg.j$\times\beta$;

rbeta.i=initlised to some random value;

rbeta.j=initlised to some random value;

rbeta.point=rbeta.i$\times g$+rbeta.j$\times\beta$;

r1=rg;

r2=r1;

next(r2,rg,rbeta);

while *true* **do**

  if *r1.point==r2.point* **then**

    if *Inverse of (r2.j-r1.j) modulo order does not exits* **then**

      Print Inverse does not exits, re-run the program;

    else

      t= Inverse of (r2.j-r1.j) modulo order ;

      α=t$\times$(r1.i-r2.i) mod order;

      Output α;

    end

  end

  next(r1,rg,rbeta);

  next(r2,rg,rbeta);

  next(r2,rg,rbeta);

end

Algorithm 1: Pollard's Rho algorithm

```
begin PROCEDURE Next
    Input: state_index t, rg, rbeta;
    Output: state_index t;
    if X-Coordinate of t.point < order /3 then
        t.point=t.point + rg.point;
        t.i=(t.i+rg.i)%order;
        t.j=(t.j+rg.j)%order;
    else if X-Coordinate of t.point < 2* order /3 then
        t.point = 2*t.point;
        t.i=(t.i+t.i)%order;
        t.j=(t.j+t.j)%order;
    else
        t.point =t.point +rbeta;
        t.i=(t.i+rbeta.i)%order;
        t.j=(t.j+rbeta.j)%order;
    end
end
```

2.5.3 Pollard lambda method

The problem is to find $\alpha g = \beta$ given g and β with additional knowledge that $L \leq \alpha \leq U$. Let $I = U - L$. We can use the Shanks algorithm to find α in time and space complexity $O(\sqrt{I})$.

We define two function f_1 and f_2 on α and g. We define f_1 and f_2 function such that they generate output randomly. One function which takes small steps is called tame kangaroo. Another function which takes large steps is called wild kangaroo. $f_1^1 = f$, $f_1^2 = f_1(f_1(g, \beta))$ and so on. $f_2^1 = f$, $f_2^2 = f_2(f_2(g, \beta))$ and so on. Example of f_1 is $f_1^1 = U \times g$ and $f_1^{i+1} = x-$coordinate of $f_1^i \times g$. Example of f_2 is $f_2^1 = \beta$ and $f_2^{i+1} = x-$coordinate of $f_2^i \times g$

We keep on computing $f_1^2, f_1^3, f_1^4, \ldots$ and $f_2^2, f_2^3, f_2^4 \ldots$. If there is match in the values generated by f_1 and f_2 we got a relationship involving g and β so we can solve for α.

2.5.4 Algorithm for Pollard's lambda method

The equation we solve is $\alpha\, g = \beta$ where α is the key which is unknown and g is the generator. g, β are called public parameters. The user gives the input of g, β, and order of the curve. The user gives the range of α i.e $L \leq \alpha \leq U$

Input: g the generator of the elliptic curve.

β the public parameter.

order : the number of points on the curve

a, b, p : Parameter that define elliptic curve $y^2 = x^3 + ax + b$ defined over Z_p

L, U: the range of α i.e $L \leq \alpha \leq U$

Output: α

The description of Pollard's Lambda algorithm

n=$U - L$;

interval=10 $\times\sqrt{n}$;

count1=0;

count2=0;

x0=b$\times$ g ;

xi=x0;

d1=f(x0);

for *i in 1 to interval* **do**

    temp=f(xi);

    xi=xi+temp $\times$ g ;

    temp=f(xi);

    d1=d1+temp;

    count1=count1+1;

end

y0=β ;

yi=y0; d2=f(y0);

while *TRUE* **do**

    temp=f(yi);

    yi=yi+temp $\times$ g;

    temp=f(yi); d2=d2+temp;

    if *yi==xi* **then**

        Print The solution is U+d1-d2 mod order;

        return *U+d1-d2 mod order* ;

    end

    if *d2 > U-L + d1* **then**

        Print The algorithm failed to find the solution. ;

        return *0* ;

    end

end

Algorithm 2: Pollard's Lambda Algorithm

```
begin PROCEDURE f
    Input: EllipticCurvePoint p;
    Output: integer t;
    /* takes a point on elliptic curve as input and return
       an integer.  If pollard's Lambda fails to find the
       key we have to change the function.                 */
    if x-coordinate of p == zero then
        return 0;
    end
    t=(x-coordinate of p) - (y-coordinate of p);
    if t <= 0 then
        return -t;
    else
        return t ;
    end
end
```

2.6 MOV Attack

The MOV attack [19], named after Menezes, Okamoto and Vanstone uses the Weil pairing to convert a discrete log problem in $E(\mathbb{F}_q)$ to one in $\mathbb{F}^*_{q^m}$. m is the embedding degree. MOV is a kind of attack that does not solve the ECDLP problem directly. It converts ECDLP into DLP. So we can use index calculus method to solve it. As index calculus is subexponential algorithm we can solve the ECDLP in subexponential time. There are no subexponential algorithms on ECDLP.

2.6.1 The MOV algorithm

1. Compute the number of points $\#E(\mathbb{F}_{p^k})$. Say N. Ord($g|N$) (since g is a point on $E(\mathbb{F}_{p^k})$.
2. Choose a random point $U \in E(\mathbb{F}_{p^k})$ with $U \notin E(\mathbb{F}_p)$.
3. Let $U' = (N/\text{Ord}(g))T$. U' is a point of order g , so proceed to Step 4.
4. Compute $A = e_{\text{ord}(g)}(g, U') \in \mathbb{F}^*_{p^k}$ and $B = e_{\text{ord}(g)}(\beta, U') \in \mathbb{F}^*_{p^k}$. Where e is Weil pairing calculated using Millers algorithm.
5. Solve the DLP for A and B in $\mathbb{F}^*_{p^k}$, i.e., find an exponent α such that $B = A^\alpha$.

6. This α is same as the α in the equation $\beta = \alpha g$, so the ECDLP has been solved.

Analysis

Let us suppose $m = 2$. Then $E(\mathbb{F}_q)$ is converted into $E(\mathbb{F}_{q^2})^*$. If q is of 160 bits then DLP will be of 160*2 bits which is quite easy to solve using index calculus method. In supersingular curves the embedding degree is low. So we have to avoid these curves.

Let us suppose $m > 6$. Then the ECDLP problem is converted into DLP problem of large size. Although we have subexponential algorithm, it takes more time as the problem size is large. So this attack is infeasible.

2.7 Libraries

For implementing various algorithms reported in this thesis, we have used the following packages

1. Sage
2. GMP

2.7.1 Sage

Sage (previously SAGE, System for Algebra and Geometry Experimentation) is a free open-source mathematics software system that builds on top of many existing open-source packages like NumPy, R, GAP, Maxima, etc. Sage is a mathematics computing environment for performing symbolic, algebraic and numerical computations. Sage is an open source alternative to Magma, Maple, Mathematica and MATLAB. William A. Stein at University of Washington, the lead developer of Sage, has integrated many specialized mathematical softwares into a common interface using Python and Cython.

Sage can be accessed as a web service using a web browser called *Sage Notebook* at `http://www.sagenb.org/` or using a command line interface. Sage development team provides a public Sage cloud service called *SageMathCloud* at (`https://cloud.sagemath.com/`).

Elliptic Curves

There are many ways to construct an elliptic curve in Sage. We can constant an elliptic curve as `EllipticCurve([a1,a2,a3,a4,a6])` where `a1, a2, a3, a4, a6` represents the curve equation in a-invariants as explained in Section 2.1(Page no:5). The elliptic curve will be defined over Rational Field

```
 e=EllipticCurve([1,2,3,4,5]);
 print e;
OUTPUT:
 Elliptic Curve defined by y^2 + x*y + 3*y = x^3 + 2*x^2 + 4*x + 5
 over Rational Field
```

`EllipticCurve([a4,a6])` will define a elliptic curve of the form $y^2 = x^3 + a_4x + a_6$ over a Rational Field. `EllipticCurve(GF(p^r),[a4,a6])` define a elliptic curve $y^2 = x^3 + a_4x + a_6$ over a finite field F_{p^r}

```
 e1=EllipticCurve(GF(13),[1,4]);
 print e1;
 f.<a>=GF(3^3);
 e2=EllipticCurve([a,a^2]);
 print e2;
OUTPUT:
 Elliptic Curve defined by y^2 = x^3 + x + 4 over
 Finite Field of size 13
 Elliptic Curve defined by y^2 = x^3 + a*x + a^2 over
 Finite Field in a of size 3^3
```

The next step after the creation of the elliptic curve is to define the points on the curve. If `e1` is the elliptic curve `e1(x1,y1)` defines the point on the elliptic curve and can be assigned to the variable `p` or `q`. We can perform the addition of points on elliptic curve as `p+q` and we can perform scalar multiplication as `n*p`.

```
 e1=EllipticCurve(GF(13),[1,4]);
 f.<a>=GF(3^3);
 e2=EllipticCurve([a,a^2]);
 p=e1(2,1);
 q=e2(1,a+2);
```

```
 r=e2.random_point();
 print p;
 print q;
 print r;
OUTPUT:
 (2 : 1 : 1)
 (1 : a + 2 : 1)
 (2*a : 2*a^2 + 2 : 1)
```

We have used Sage in implementing the proposed Two key ECDSA which will explained in the next chapter. Sample code of Two key ECDSA implementation is given in Appendix

2.7.2 GMP

GMP, acronym for GNU Multiple Precision Arithmetic Library, is a portable library for handling arbitrary precision arithmetic [14]. Although there are many way to install GMP, the easiest way to install GMP is by using Software Center or Add/Remove program manager in Linux environment.

GMP installation gives us a gmp header file. All GMP programs should include the header file gmp.h which provides the declaration for a number of data structures and functions defined in the library.

```
#include<gmp.h>
```

We can declare the GMP multiple precision integer as

```
mpz_t b;
```

in the declaration section of a C program or anywhere in a C++ program. Like the pointer which needs to be initialised with `malloc` or `new` before using it, we need to initialise the `mpz_t` variable to use it. For allocating memory we use `mpz_init`, which also sets it value to zero.

```
mpz_init(b);
```

`mpz_init_set_si(b,t)` the function allocates memory and assigns the value of `t` to `b`. Here `si` indicates that the variable `t` is of type signed integer.

We want to assign a 11-digit or more number to the `mpz_t` variable.
If we use the `mpz_init_set_si` the 11-digit or more number cannot be assigned because the number is first converted to int data type in C and then assigns to `mpz_t`. While complilation it gives a warning also.
`mpz_init_set_str(gmp-integer,"string",base)` allocates memory to `gmp-integer` and assigns the value of `string` treating it as the `base` integer.
`mpz_set_si(b,op2)` Assigns the value of `op2` to gmp-integer `b`. Here op2 is of type `int`. Here it is assumed that `b` has been allocated memory by some function. If memory is not allocated to `b` it results in run time error. All the functions except function which has init word in its name assume that the memory has been allocated.
`mpz_set(b1,b2)` Assigns the value of `b2` to `b1`. Here b1 and b2 are of type gmp-integers and are allocated memory.

`mpz_add(result,op1,op2)`, `mpz_sub(result,op1,op2)`,
`mpz_mul(result,op1,op2)` These function assign `result` $\leftarrow$ `op1 + op2`, `result` $\leftarrow$ `op1 - op2`, `result` $\leftarrow$ `op1 * op2` respectively. Here result, op1 and op2 are of type `mpz_t` and are allocated memory.

`mpz_fdiv_q(result,number,divisor)` This function divides the `number` by `divisor` and stores the quotient in the variable `result`.
`mpz_fdiv_qr(result,rem,number,divisor)` This function divides the `number` by `divisor` and stores the quotient in the variable `result` and stores the remainder in the variable `rem`.
`mpz_mod(result,number,mod)` This function divides the `number` by `mod` and stores the remainder in the variable `result`.
`mpz_cmp(op1,op2)` This function compares op1 and op2, return a positive integer if op1 $>$ op2, returns zero if op1==op2, return a negative integer if op1 $<$ op2.
`gmp_printf` This function is similar to printf function in C and accepts all the formatting strings of printf. It also accepts %Zd format for printing `mpz_t` variable.
`gmp_scanf` This function is similar to scanf function in C and accepts all the formatting strings of printf. It also accepts %Zd format for scanning `mpz_t` variable.
`mpz_clear(b)` This function releases the memory allocated to the variable `b`.

We have used the GMP library in implementing the function, briefly described in Appendix, which in turn are used for implementation of the existing Gopalakrishnan et al. and our proposed schemes. We have executed the implemented programs of Gopalakrishnan et al. and our proposed schemes and tabulated the results in Chapter 5.

In this Chapter we have introduced Arithmetic of Elliptic Curves, Elliptic Curve Discrete Logarithm Problem(ECDLP) and schemes based on ECDLP. These topics provide necessary background for the Chapter 3 (Two key Elliptic Curve Digital Signature Algo-

rithm). In Chapter 3 we will be proposing a new scheme. In this Chapter we have also introduced Algorithms for solving discrete log problem and libraries used for implementing these algorithms. In Chapter 4 (Solving discrete logarithms from partial knowledge of the key) we will be describing the algorithms of Gopalakrishnan et al. which uses Pollard lambda methods.

Chapter 3

Two key Elliptic Curve Digital Signature Algorithm

This chapter describes ECDSA published in ANSCI X9.62 and our proposed Two key Elliptic curve Digital Signature algorithm. Some application of the proposed scheme are also discussed. This chapter also describes an algorithm for attacking the proposed Two key signature scheme.

3.1 Elliptic Curve Digital Signature Algorithm

The Elliptic Curve Digital Signature Algorithm (ECDSA) is the elliptic curve analogue of the Digital Signature Algorithm (DSA). It is the most widely standardized elliptic curve-based signature scheme, appearing in the ANSI X9.62, FIPS 186-2, IEEE 1363-2000 and ISO/IEC 15946-2 standards as well as several draft standards [3, 12].

The Domain parameters $D = (q, FR, S, a, b, P, n, h)$ comprise

1. The field order q.
2. An indication FR (field representation) of the representation used for the elements of $\mathbb{F}_q$.
3. A seed S if the elliptic curve was randomly generated.
4. Two coefficients a, $b \in \mathbb{F}_q$ that define the equation of the elliptic curve E over $\mathbb{F}_q$ (i.e $y^2 = x^3 + a\,x + b$ in the case of a prime field and $y^2 + xy = x^3 + a\,x^2 + b$ in case of binary field).

5. Two field elements x_p and y_p in $\mathbb{F}_q$ that define a finite point $P = (x_p, y_p) \in E(\mathbb{F}_q)$ in affine coordinates. P is called the base point.

6. The order n of P.

7. The cofactor $h = \#E(\mathbb{F}_q)/n$.

In the following, H denotes a cryptographic hash function whose outputs have bit length no more than that of n. If this condition is not satisfied, then the outputs of H can be truncated.

3.1.1 ECDSA Signature generation

Input: Domain parameters $D = (q, FR, S, a, b, P, n, h)$, private key d, message m.
Output: Signature (r, s)
begin

1 Select $k \in_R [1, n-1]$;
2 Compute $kP = (x_1, y_1)$ and convert x_1 to an integer $\bar{x}_1$;
3 Compute $r = \bar{x}_1 \bmod n$;
4 **if** $r = 0$ **then**
  goto step 1
  end
5 Compute $e = H(m)$;
6 Compute $s = k^{-1}(e + d\,r) \bmod n$;
7 **if** $s = 0$ **then**
  goto step 1
  end
8 **return** (r, s);

end

Algorithm 3: ECDSA Signature generation

3.1.2 ECDSA signature Verification

Input: Domain parameters $D = (q, FR, S, a, b, P, n, h)$, public key Q, message m, signature (r, s).

Output: Acceptance or Rejection of the signature

begin

Verify that r and s are integers in the interval $[1, n-1]$. If verification fails then

return *(Reject the signature)* ;

Compute $e = H(m)$;

Compute w = s^{-1} mod n ;

Compute $u_1 = ew$ mod n, $u_2 = wr$ mod n ;

Compute $X = u_1P + u_2Q.$;

if X = $\mathcal{O}$ **then**

  return *(Reject the signature)*

end

Convert the x-coordinate x_1 of X to an integer $\bar{x}_1$;

Compute $v = \bar{x}_1$ mod $n.$;

if $v = r$ **then**

  return *(Accept the signature)*

else

  return *(Reject the signature)*

end

end

Algorithm 4: ECDSA signature Verification

3.1.3 Proof that signature verification works

If a signature $(r,\ s)$ on a message m was indeed generated by the legitimate signer, then $s \equiv k^{-1}(e + d\,r) \pmod{n}$. Rearranging gives

$$k \equiv s^{-1}(e + d\,r) \equiv s^{-1}e + s^{-1}rd \equiv we + wrd \equiv u_1 + u_2d \pmod{n}.$$

Thus $X = u_1P + u_2Q = (u_1 + u_2d)P = kP$, and so $v = r$ as required.

In ECDSA we have only one key. There are many applications which require more than one person to sign a document (agreement/contract) to make it valid. There are security scenarios involving a group of people who all need to be present for some secret to be decrypted. If we have a signature scheme with several keys we can distribute one key to each person. To sign a document all members individually sign the document with their

keys and combine their signature using some function to generate the combined signature. We want to study if ECDSA can be generalised into an N-Key ECDSA and, in particular, Two-Key ECDSA.

3.2 Two Key ECDSA

The domain parameters $D = (q, FR, S, a, b, P, P_1, P_2Q, R, n, h)$ are same as the original ECDSA except P_1, P_2 and R. P_1 and P_2 are two base point.

Let d_1 and d_2 be the private keys and d_1P_1 + d_2P_2 =R and $dP = Q$

P, P_1, P_2 and R are the public parameters.

H defines the hash function as in ECDSA.

3.2.1 Two Key ECDSA Signature Generation

Input: Domain parameters $D = (q, FR, S, a, b, P_1, P_2R, n, h)$, private key d_1 and d_2, message m.
Output: Signature (x_1, s_1, s_2)
begin

1. Select k_1 and $k_2 \in_R [1, n-1]$;
2. Compute k_1P_1 + k_2P_2 = $S(x_1, y_1)$;
3. **if** S=$\mathcal{O}$ **then**
   goto step 1
   end
4. Compute $e = H(m)$.;
5. Convert the field element x_1 to an integer x (usually the first coordinate in the vector representation) ;
6. Compute $s_1 = e\,k_1 + x\,d_1$ and $s_2 = e\,k_2 + x\,d_2$;
7. **return** *(x_1, s_1, s_2)*;

end

Algorithm 5: Two Key ECDSA Signature Generation

3.2.2 Two key ECDSA Signature Verification

Input: Domain parameters $D = (q, FR, S, a, b, P_1, P_2R, n, h)$, public key P, Q, P_1, R message m, signature (x_1, s_1, s_2)

Output: Accept or Reject the signature

begin

Compute $e = H(m)$;

Compute $u_1 = s_1P_1 + s_2P_2$;

Compute y_1 and y_1' from x_1 by curve equation ;

Let $T = (x_1, y_1)$ and $T' = (x_1, y_1')$;

Compute $u_2 = e\,T + x_1R$;

if $u_1 = u_2$ **then**

  return *(Accept the signature)*

else if $u_1 = e\,T' + x_1R$ **then**

  return *(Accept the signature)*

else

  return *(Reject the signature)*

end

end

Algorithm 6: Two key ECDSA Signature Verification

3.2.3 Proof that the signature verification works

If a signature (x_1, s_1, s_2) on a message m was indeed generated by the legitimate signer, then

$$\begin{aligned} s_1P_1 + s_2P_2 &= (e\,k_1 + x_1d_1)P_1 + (e\,k_2 + x_1d_2)P_2 \\ &= e(k_1P_1 + k_2P_2) + x_1(d_1P_1 + d_2P_2) \\ &= e\,T + x_1R \end{aligned}$$

3.2.4 Analysis of Two key ECDSA

As the equation to solve is $d_1P_1 + d_2P_2 = R$ we cannot apply Pohlig-Hellman algorithm. So we consider the brute force as the attack model. We choose an elliptic curve such that the curve has two independent generators. So the generators P_1 and P_2 form the basis for the elliptic curve.

Proposition 1 *There will be unique values of d_1 and d_2 satisfying the equation $d_1P_1 + d_2P_2 = R$ where P_1 and P_2 are both independent generators of the elliptic curve.* □

PROOF We prove this by contradiction.

Assume that there are two other values u_1 and u_2 which satisfy the equation.

So $u_1P_1 + u_2P_2 = R$. But we know $d_1P_1 + d_2P_2 = R$,
$u_1P_1 + u_2P_2 = d_1P_1 + d_2P_2$
$(u_1 - d_1)P_1 = (d_2 - u_2)P_2$

Which contradicts our assumption that P_1 and P_2 are independent generators. So there will be unique values of d_1 and d_2 satisfying the equation $d_1P_1 + d_2P_2 = R$. ∎

Modified Shanks algorithm

We try to solve the equation $d_1P_1 + d_2P_2 = R$. So we write the equation as

$$d_1 \times P_1 = R - d_2 \times P_2$$

We compute $R - i \times P_2$ for i varying from 0 to n and store the point and i in the table . Then we compute $P_1, 2P_1 \ldots$ and keep checking in the table. If match is found we calculate d_1 and d_2 as follows

$R - x'P_2 = y \times P_1$ rearranging gives
$R = y \times P_1 + x'P_2$.
So y and x' are d_1 and d_2 respectively.

Input: P_1, P_2, R along with the equation of the curve and the field.
Output: d_1, d_2 with satisfy the equation $d_1P_1 + d_2P_2 = R$.
begin
  table = ϕ ;
  for i *in* 0 *to* n **do**
    $R' = R - i \times P_2$;
    table=table $\cup\{(R', i)\}$;
  end
  for j *in* 0 *to* n_1 **do**
    P' = $j \times P$;
    search linearly in the table for P' ;
    if *match found* **then**
      $d_1 = j$;
      $d_2 = i$;
      return d_1, d_2 ;
    end
  end
end

Algorithm 7: Algorithm to solve Two Key ECDSA problem

The time complexity of the algorithm is $O(n_1)$+$O(n_1) = O(n_1$+$n_2) = O(2$∗$n) = O(n)$ (assuming that n, n_1 are approximately equal).

3.2.5 Comparison of ECDSA and Two key ECDSA

If we want to sign a message with one key we can use standard ECDSA with the domain parameters of Two key ECDSA. The person who is signing can decide whether to use two keys or one key.

The brute force method to attack the ECDSA is to find d (private key) from public key Q and domain parameter P which satisfy the relationship $Q = d\,P$. The order of P is n so to find Q we have to check n possibilities. So the time complexity is O(n). Using the Modified Shanks algorithm described above the time complexity to break the Two key signature scheme is O(n). The security the two key signature offers is $O(n)$. We have doubled the key size and the security increased linearly. Breaking of the Two key algorithm takes twice the time needed to break the single key ECDSA. That is, the security increases linearly in the number of bits of the keys (as two keys contain $2n$ bits) while in cryptography we look for exponential increase in security in the number of bits. Thus, extending the ECDSA scheme with multiple keys does not give as much additional security as expected.

However, such multiple keys signature schemes are useful in certain applications.

3.3 Application in Counter Signing Documents

Let us suppose there are two partners in a business. There is an agreement between them that if both of them sign a document, then only the document is valid. This kind of scenarios typically exist in banks. We can realise this using the proposed Two key signature scheme. Here we have two signers. The public parameters are given to both the signers. One private key is given to one signer and other private key is given to another signer.
The signing algorithm will be as follows for signer 1
Signer1 generates random numbers k_1 and k_2 and computes $k_1\,P + k_2\,P_1 = S$. He calculates hash e and computes s_1 as $ek_1 + xd$. Signer 1 Publishes his signature (x_1,s_1) and communicates the k_2 secretly to Signer 2. Signer 2 calculates $s_2 = ek_2 + xd_1$ and publish s_2 as his signature. Total signature will be the combination of Signer 1 signature and Signer 2 signature.

3.3.1 Algorithm for Signer 1

Input: Domain parameters $D = (q, FR, S, a, b, P_1, P_2, R, n, h)$, private key d_1, message m
Output: Signature (x_1, s_1)
begin
1. Select k_1 and $k_2 \in_R [1, n-1]$;
2. Compute $k_1P_1 + k_2P_2 = S(x_1, y_1)$. ;
3. **if** $S = \mathcal{O}$ **then**
   goto step 1
   end
4. Secretly communicate k_2 to Signer 2 ;
5. Compute e=H(m);
6. Convert the field element x_1 to an integer x ;
7. Let $s_1 = ek_1 + xd_1$;
8. **return** *Signer 1 signature* (x_1, s_1)

end

Algorithm 8: Algorithm for Signer 1

3.3.2 Algorithm for Signer 2

Input: Domain parameters $D = (q, FR, S, a, b, P_1, P_2, R, n, h)$, private key d_2, message m, Signers 1 signature (x_1, s_1), secretly communicated k_2

Output: Signature (x_1, s_1, s_2)

begin

Compute e=H(m) ;

Convert the field element x_1 to an integer x ;

Compute $s_2 = e\,k_2 + x\,d_2$;

return (x_1, s_1, s_2) as the combined signature;

end

Algorithm 9: Algorithm for Signer 2

The Algorithm 6 described in page 29 can be used to verify the signatures.

3.4 Comparison of both schemes with examples

Example 1

The elliptic curve is defined by the following parameters:
The prime p = 252151, a = 337, b = 47
The base points P_1 =(184870,17195), P_2 = (129882,52865).
The order of the curve is 252054.
The order of P_1 is 84018, order of P_2 is 3
Signature generation
The private keys d1= 20 d2= 41
P1= (184870 : 17195 : 1) P2= (129882 : 52865 : 1) R= (69369 : 149658 : 1)
The random numbers selected are k1= 5 k2= 8
S= (237658 : 101436 : 1)
Hash of the message is= 28
Signature is 237658, 216328, 166150
Signature verification
u1= (154209 : 95062 : 1)
T= (237658 : 101436 : 1) T1= (237658 : 150715 : 1)
u2= (154209 : 95062 : 1)
As u1=u2,
Accept the signature
The time complexity for breaking the signature is order of P + order of Q.

So the number of computations required to break this two key signature scheme are $84018 + 3 = 84021$ (approx).
Whereas the security of standard / one key ECDSA is order of P. Number of computations required to break standard / one key ECDSA are 84018 (approx).

Example 2

The elliptic curve is defined by the following parameters:
The prime p=1987, a=1, b=152, Finite field is GF(p^2)
The base points are P_1= (480*x + 1579, 736*x + 82), P_2=(201*x + 1566, 568*x + 739)
The order of the curve is 3952144.
The order of P_1 is 1988, order of P_2 is 1988
Signature generation
The private keys d1= 20 d2= 41
P1= (710*x + 1299 : 551*x + 182 : 1) P2= (1104*x + 85 : 591*x + 1473 : 1)
R= (1103*x + 1152 : 1634*x + 628 : 1)
The random numbers selected are k1= 5 k2= 8
S= (14*x + 1570 : 1311*x + 87 : 1)
Hash of the message is= 28
Signature is 14*x + 1570, 31540, 64594
————————signature generated————-
u1= (1824*x + 801 : 1462*x + 845 : 1)
T= (14*x + 1570 : 676*x + 1900 : 1) T1= (14*x + 1570 : 1311*x + 87 : 1)
u2= (1237*x + 1982 : 718*x + 281 : 1) calculated using T
u2= (1824*x + 801 : 1462*x + 845 : 1) calculated using T1
Accept the signature
The time for breaking the signature is order of P1 + order of P2
So the number of computations required to break this two key signature scheme are $1988 + 1988 = 3976$ (approx).
Whereas the security of standard / one key ECDSA is order of P. Number of computations required to break standard / one key ECDSA are 1988 (approx).

3.5 Application in Digital Certificates

Deffi-Hellman Key exchange designed to solve the problem of key exchange. But it suffer the problem of man-in-the-middle attack. The problem can be solved by using digital certificates. Digital Certificate is a computer file with encoding as in described in [7]. The certificate is described in ASN.1 (Abstract Syntax Notation One) as

```
Certificate  ::=  SEQUENCE  {
      tbsCertificate       TBSCertificate,
      signatureAlgorithm   AlgorithmIdentifier,
      signatureValue       BIT STRING
      }

TBSCertificate  ::=  SEQUENCE  {
      version         [0]  EXPLICIT Version DEFAULT v1,
      serialNumber         CertificateSerialNumber,
      signature            AlgorithmIdentifier,
      issuer               Name,
      validity             Validity,
      subject              Name,
      subjectPublicKeyInfo SubjectPublicKeyInfo,
      issuerUniqueID  [1]  IMPLICIT UniqueIdentifier OPTIONAL,
                           -- If present, version MUST be v2 or v3
      subjectUniqueID [2]  IMPLICIT UniqueIdentifier OPTIONAL,
                           -- If present, version MUST be v2 or v3
      extensions      [3]  EXPLICIT Extensions OPTIONAL
                           -- If present, version MUST be v3
      }

 Version  ::=  INTEGER  {  v1(0), v2(1), v3(2)  }

 CertificateSerialNumber  ::=  INTEGER

 Validity ::= SEQUENCE {
      notBefore      Time,
      notAfter       Time
      }

 Time ::= CHOICE {
```

```
        utcTime         UTCTime,
        generalTime     GeneralizedTime
        }

UniqueIdentifier  ::=  BIT STRING

SubjectPublicKeyInfo  ::=  SEQUENCE  {
        algorithm            AlgorithmIdentifier,
        subjectPublicKey     BIT STRING
        }


Extensions  ::=  SEQUENCE SIZE (1..MAX) OF Extension


Extension  ::=  SEQUENCE  {
        extnID      OBJECT IDENTIFIER,
        critical    BOOLEAN DEFAULT FALSE,
        extnValue   OCTET STRING
        }
```

The format is encoded using Distinguished Encoding Rules (DER). The subjectPublicKey gives the information of the public keys and all the domain parameters of the two key signature scheme. A new value for AlgorithmIdentifier is identified.

3.5.1 Usage

As Two key ECDSA is backward compatible, subset of domain parameters of Two key ECDSA are required to use One key ECDSA. The user can use either one key or two key ECDSA algorithms depending on the security requirements for signing the documents, SSL encryption, etc.

This chapter describes about ECDSA published in ANSCI X9.62 and our proposed Two Key ECDSA and some applications and cryptanalysis of the proposed scheme. This chapter can be treated as a contribution in the area of cryptography. Chapter 5 (Efficient Algorithm to Solve DLP from partial knowledge of the key) describes an attacks on the ECDLP problem. Next chapter can be treated as a contribution in the area of cryptanalysis.

Chapter 4

Solving discrete logarithms from partial knowledge of the key

This chapter contains the work of Gopalakrishna et al. [16] in detail so that our work in the next chapter can be fully understood.

4.1 Partial Knowledge of the key

We know that key is nothing but the combination of zeros and ones. Partial knowledge of the key of the key means that we know some portions/region of the key (one or more). We may know the position of those regions or we may not know. The problem of solving discrete logarithms from partial knowledge of the key can be defined as find the secret key using the partial knowledge and public parameters efficiently. Depending on these possibilities we can classify the partial knowledge of key into these categories.

1. We know one portion of the key along with the position where it is occurring in the key.
   Eg: Let the key be 100010010 which is secret. We know that 100 is in the key at position 5 from MSB (Most Significant Bit), which can be called as partial knowledge of the key.

2. We know more than one portion of the key along with the positions where they are occurring in the key.
   Eg: Let the key be 1011 1010 0010 which is secret. Here we know that 110 is in the key at position 4 from MSB and 001 is in the key at position 9 from MSB.

3. We know one portion of the key along without the position where it is occurring in the key
   Eg: Let the key be 100010010 which is secret. We know that 100 is in the key, but do not know its position in the key.

4. We know more than one portion of the key along with the positions where they are occurring in the key.
   Eg: Let the key be 1011 1010 0010 which is secret. Here we know that 110, 001 is in the key, but do not know their position in the key.

5. We know more than one portion of the key along with the positions of some portions, but not all portions where they are occurring in the key.
   Eg: Let the key be 1011 1010 0010 which is secret. Here we know that 110, 001 is in the key We also know that 110 is at position 4. We do not know the position of 001.

In this chapter we will present the work of Gopalakrishna et al. [16] which is mainly the contribution when partial knowledge available belongs to Category 1 described above. In the next chapter we will present our work which is mainly the contribution when partial knowledge available belongs to Category 2 described above.

4.2 Notation

Let G be a cyclic group. Let $g \in G$ of order Ord(g). Given β an element in the sub group generated by g determine $\alpha \in \{1, 2, \ldots, \text{Ord}(g)\}$ such that $\alpha\, g = \beta$. Here β and g are public parameters and the secret key is α. So DLP can be defined as described in Section 2.2 (Page No. 9). From partial knowledge we try to solve the DLP problem in this Chapter and the next Chapter assuming only the group structure of the elliptic curve, means we do not assume the prime number concept. So the algorithm presented in this Chapter and next Chapter can be used on both DLP and ECDLP problems. Let l be the number of bits of α represented in binary.

4.3 Solving DLP from Partial knowledge of key

Gopalakrishna et al. [16] has presented an algorithm that improves on the exhaustive search when the partial information of the key along with position is revealed i.e case 1 in the above classification. Again the Case 1 has been divided into three sub cases.

1. Left part of the key is revealed.
2. Right part of the key is revealed.
3. Middle part of the key is revealed.

4.3.1 Left part of the key is revealed

We assume that most significant bits of the key are known.
So the key will look like (We use || symbol for concatenation)
α= known bits || unknown bits
As we know the length of the key (in binary) and length of the known bits (in binary) we know the length of unknown bits (in binary). We replace all the unknown bits by zero. This becomes the smallest possible value for α let it be a.
a = known bits || $0000\ldots00$.
We replace all the unknown bits by 1.
This becomes the largest possible value for α, let it be b.
b=known bits || $1111\ldots00$.
We know $a \leq \alpha \leq b$. We can use Pollard's Kangaroo Algorithm with expected running time of $O(\sqrt{b-a})$

4.3.2 Right part of the key is revealed

We assume that contiguous least significant bits of the key are known. Let α_2 denote the known contiguous bits. Suppose that l is the length of the key. Suppose that l_2 is the length of the known sequence of contiguous bits and l_1 is the length of the remaining bits so that $l = l_1 + l_2$.

So the key will look like
α = unknown bits || known bits = $\alpha_1||\alpha_2$.
So we can write

$$\alpha = \alpha_1 \times 2^{l_2} + \alpha_2$$

Let $M = \lfloor \frac{p-\alpha_2-1}{2^{l_2}} \rfloor$, then we know that $0 \leq \alpha_1 \leq M$ since $0 \leq \alpha \leq p-1$.

$$\begin{aligned}\beta = \alpha\, g &= (\alpha_1 \times 2^{l_2} + \alpha_2)g \\ &= (g \times 2^{l_2})\alpha_1 + g \times \alpha_2\end{aligned}$$

If we denote $2^{l_2}\ g$ by g' and $\beta - \alpha_2 g$ by β' the above equation reduces to $\beta' = \alpha_1 g'$. We can then solve this DLP by using Pollard's Kangaroo Algorithm on g' and β' in $O(\sqrt{M})$. As $M = \lfloor \frac{p-\alpha_2-1}{2^{l_2}} \rfloor$, the complexity is easily seen to be $O(\sqrt{\frac{p}{2^{l_2}}})$

Example 1 Let the key be 1010110. We know only least significant two bits i.e 10. So the known key is 10 (α_2) and the unknown key is 10101 (α_1). So we can write (We use $||$ symbol for concatenation)
$\alpha = 10101||10$ as the know bit length is 2 we can write
$\alpha = 10101 \times 2^2 + 10$
$\alpha = \alpha_1 \times 2^2 + 10$
We know $\alpha g = \beta$
$\alpha_1 \times 2^2 + 10)g = \beta$
Rearranging gives $\alpha_1(2^2 \times g) = (\beta - 10g)$
As we know all the terms except α_1 we can rewrite the equation as
$\alpha_1 g' = \beta'$ where $g' = 2^2 \times g$ and $\beta' = \beta - 10g$
$\alpha_1 g' = \beta'$ is another DLP problem, so we can use Pollard's Kangaroo Algorithm to solve for α_1 □

4.3.3 Middle part of the key is revealed

Let α be the key. Let α_1 and α_3 are unknown and α_2 is known.

We have positive integers M and N such that we can write α in the form

$$\alpha = \alpha_1 MN + \alpha_2 M + \alpha_3 \tag{4.1}$$

We assume that $0 \leq \alpha_1 < p/MN$, i.e. that α is reduced modulo p. Note that α_1 is really bounded by $\lfloor \frac{p-\alpha_2 M}{MN} \rfloor$

Let p (prime) be the order of g. Let r be an integer $0 < r < p$ such that rMN as $kp + s$ with $|s| < p/2$.

$$\begin{aligned} r\alpha &= r\alpha_1 MN + r\alpha_2 M + r\alpha_3 \\ &= \alpha_1 kp + s\alpha_1 + r\alpha_2 M + r\alpha_3 \\ &= \alpha_1 kp + r\alpha_2 M + \alpha' \end{aligned}$$

where α' = $s\alpha_1 + r\alpha_3$. Multiplying g to both sides we get

$$\begin{aligned} g \times r\alpha &= g \times (\alpha_1 kp + r\alpha_2 M + \alpha') \\ (g\alpha)\, r &= (gp)\alpha_1 k + gr\alpha_2 M + g\alpha' \\ \beta r &= gr\alpha_2 M + g\alpha' \\ \beta' &= g\alpha' \text{ where } \beta' = (\beta - g\alpha_2 M)\, r \end{aligned}$$

The interval of α' is $\left[0, r(M-1) + s\left(\frac{p}{MN} - 1\right)\right]$ when $s > 0$. If s is negative we must consider the interval $\left[s\left(\frac{p}{MN} - 1\right), r(M-1)\right]$. In both cases we can restrict the value of α to an interval of length $rM + |s|\frac{p}{MN}$. The time complexity is $O(\sqrt{2p}/N^{1/4})$. From $\alpha' = r\alpha_3 + s\alpha_1$, we have to solve an easy diophantine equation to obtain α_1 and α_3 to get α from (4.1)

Example 2 Let the key be 1101100010. The known bits are 1100 the middle part.
So $\alpha_1 = 110$, $\alpha_2 = 1100$ and $\alpha_3 = 010$. No. of bits of α_1 = 3,
No. of bits of α_2 = 4,
No. of bits of α_3 = 3.
The known part is α_2 and the unknown parts of the key are α_1 and α_3. So $\alpha = 110||1100||010$
$\alpha = \alpha_1||1100 \times 2^{\text{no: of bits of } \alpha_3} + \alpha_3$
$\alpha = \alpha_1||1100 \times 2^3 + \alpha_3$
$\alpha = \alpha_1 \times 2^3 \times 2^{\text{no: of bits of } \alpha_1} + 1100 \times 2^3 + \alpha_3$
$\alpha = \alpha_1 \times 2^3 \times 2^4 + 1100 \times 2^3 + \alpha_3$. Let $2^3 = M$ and $2^4 = N$
$\alpha = \alpha_1 MN + 1100 \times M + \alpha_3$

Solving the Diophantine equation

To find α we have to solve the equation $\alpha' = r\alpha_3 + s\alpha_1$ and obtain α_1 and α_3, then use them in (4.1).

We know $s = rMN - kp$ as p is prime and M and N are powers of 2. If gcd$(k, r) = d > 1$. we replace $\tilde{r} = r/d$ and $\tilde{k} = k/d$, which gives us $\tilde{s} = \tilde{r}MN - \tilde{k}p = s/d$. So $\tilde{s} = \tilde{r}MN - \tilde{k}p$ gives

$$\left|\frac{MN}{p} - \frac{\tilde{k}}{\tilde{r}}\right| < \frac{(MNT/p^2)}{\tilde{r}}$$

r is coprime to p since p is a prime.

$$\gcd(\tilde{r}, \tilde{s}) = \gcd(\tilde{r}, \tilde{r}MN - \tilde{k}p) = \gcd(\tilde{r}, \tilde{k}p) = 1.$$

We have to find all the integral solutions of $\alpha' = r\alpha_3 + s\alpha_1$. From number theory we can find the solutions as

$$\alpha_1 = b + ir$$
$$\alpha_3 = \frac{\alpha' - sb}{r} - is$$

where $b \equiv \alpha' s^{-1} \mod r$. Now we want calculate how many possible solutions are give by this equation. We know $|s| < p/2$ and $r > \frac{p}{2MN}$. So the number of possible solutions are two. We can construct α by (as α_2 is known) $\alpha = \alpha_1 MN + \alpha_2 M + \alpha_3$. We can check which solution is correct by substituting in equation $\alpha g = \beta$

By using [16] we get the time complexity as $O(\sqrt{2p}/N^{\frac{1}{4}})$ and the memory requirements will be $O(\log p)$ when we know partial information about the key. If we apply the pollard rho or pollard kangaroo method directly we get the time complexity as $O(\sqrt{n})$.

Chapter 5

Efficient Algorithm to Solving Discrete Logarithms from Partial Knowledge of the Key

In the previous chapter we have presented Gopalakrishnan et al. [16] in detail. In this Chapter we first discuss the error we found in the Gopalakrishnan et al. algorithm and suggest the possible modification to it. Then we discuss about our approach in solving the ECDLP from partial knowledge of the key. Then we describe how to solve the DLP when two portions of the key are revealed. Then we briefly describe how to solve when three portions of the key are revealed. We discuss about the analysis of the proposed algorithm theoretically and empirically. Then we describe about the parallel version of the proposed algorithm.

5.1 Our observation on Gopalakrishnan et al. algorithm

The method suggested by Gopalakrishnan et al. [16] to extract α_3 and α_1 from α' may not give the solution as it does not take into account all possible solutions for α_3 and α_1. So we suggest the following procedure.

As α' is an element of Z_p the exact value of α' can be $\alpha' + j\,p$ where j can take integer values like $0, 1, 2, \ldots$. We have to observe that the equation $\alpha' \equiv (\alpha_3 r + \alpha_1 s) \mod p$ is a congruence. So the equation can be written as $\alpha' + j\,p = (\alpha_3 r + \alpha_1 s)$

Now the parametric equations can be written as

$$\alpha_1 = b + i\,r$$

$$\alpha_3 = \frac{\alpha' - sb}{r} - is$$

where $b \equiv \alpha' s^{-1} + j\,p \mod r$. For different values of i and j we will get different α_1 and α_3. For each value we can construct α by (as α_2 is known)

$$\alpha = \alpha_1 MN + \alpha_2 M + \alpha_3$$

We will check whether $\alpha g = \beta$ or not, for some value of α if the equation satisfies then that is the required key.

Example 3 Let i be an element of $\mathbb{Z}_1 1$. Let $i = 8$. Now we want the value of i mod 5. So $i = 3$ mod 5.

In some calculation we got the i value as 19. So i mod 5 is 4, which is wrong. So $19 + j * 11$ mod 5 gives the correct values. □

5.2 Two parts of the key are revealed

We assume that two portions/parts of the key α are revealed. So the key is divided into five parts. We assume that α_2 and α_4 are known and α_1, and α_3, α_5 are unknown. M_1, M_2, M_3, M_4 and M_5 are positive integers such that the following equation holds. (Usually M_1, M_2,.. are in the form 2^{l_1}).

$$\alpha = \alpha_1 M_1 + \alpha_2 M_2 + \alpha_3 M_3 + \alpha_4 M_4 + \alpha_5 M_5$$

We know α_2 and α_4 substituting in the equation $\alpha g = \beta$ we get

$\alpha_1 M_1 g + \alpha_3 M_3 g + \alpha_5 M_5 g$ = β_1 where $\beta_1 = \beta - \alpha_2 M_2 g - \alpha_4 M_4 g$ Here $M_5 = 1$ if we take α_5 as least significant bits.

5.3 Solution by generating equation

Substituting different values for α_1, α_3, α_5 as we know the number of the bits of α_1, α_3, α_5 we generate random values such as α_{11},α_{31}, α_{51} and a_1 whose bit length is less than or equal to the max bit length of α_1, α_3, α_5. We get a set of equations such as

$$\alpha_{1i} M_1 g + \alpha_{3i} M_3 g + \alpha_{5i} M_5 g - a_i \beta_1 = R_i \tag{5.1}$$

We will substitute the generated random values and get different R_i. If we get two equations with the same R_i we can solve them for α.

Lemma 1 *There exist nontrivial values of α_{1i}, α_{3i}, α_{5i} and a_i for which LHS of Eqn 5.1 is $\mathcal{O}$*

PROOF We know $\alpha g = \beta$ and $\alpha = \alpha_1 M_1 + \alpha_2 M_2 + \alpha_3 M_3 + \alpha_4 M_4 + \alpha_5 M_5$
So $(\alpha_1 M_1 + \alpha_2 M_2 + \alpha_3 M_3 + \alpha_4 M_4 + \alpha_5 M_5)g = \beta$
$\alpha_1 M_1 g + \alpha_3 M_3 g + \alpha_5 M_5 g = \beta - \alpha_2 M_2 g - \alpha_4 M_4 P$
$\alpha_1 M_1 g + \alpha_3 M_3 g + \alpha_5 M_5 g = \beta_1$
So there exists α_1, α_3 and α_5 such that LHS of Eqn 5.1 is $\mathcal{O}$ ∎

Corollary 1 *There exist nontrivial values of α_{1i}, α_{3i}, α_{5i}, ..., $\alpha_{(2N+1)i}$ and a_i such that*

$$\alpha_{1i} M_1 g + \alpha_{3i} M_3 g + \alpha_{5i} M_5 g \ldots + \alpha_{(2n+1)i} M_{2n+1} g - a_i \beta_1 = \mathcal{O}$$ □

Lemma 2 *There exist non trivial values α_{1j}, α_{3j}, α_{5j}, α_{1k}, α_{3k} and α_{5k} such that*

$$\alpha_{1j} M_1 g + \alpha_{3j} M_3 g + \alpha_{5j} M_5 g + a_j \beta_1 = \alpha_{1k} M_1 g + \alpha_{3k} M_3 g + \alpha_{5k} M_5 g + a_k \beta_1 \quad (5.2)$$ □

for some j and k.

PROOF As α_{11}, α_{12}, ...,α_{31}, α_{32}, ...,α_{51}, α_{52}, ... are randomly generated we can assume without loss of generality for some j
$\alpha_{1j} > \alpha_1$, $\alpha_{3j} > \alpha_3$, $\alpha_{5j} > \alpha_5$,
So let $\alpha_{1k} = \alpha_{1j} - \alpha_1$, $\alpha_{3k} = \alpha_{3j} - \alpha_3$, $\alpha_{5k} = \alpha_{5j} - \alpha_5$. Let $a_k = 2$
$\alpha_{1k} M_1 g + \alpha_{3k} M_3 g + \alpha_{5k} M_5 g + a_k \beta_1$

$= \alpha_{1j} M_1 g - \alpha_1 M_1 g + \alpha_{3j} M_3 g - \alpha_3 M_3 g + \alpha_{5j} M_5 g - \alpha_5 M_5 g + a_k \beta_1$

$= \alpha_{1j} M_1 g + \alpha_{3j} M_3 g + \alpha_{5j} M_5 g + 2\beta_1 - (\alpha_1 M_1 g + \alpha_3 M_3 g + \alpha_5 M_5 g)$

$= \alpha_{1j} M_1 g + \alpha_{3j} M_3 g + \alpha_{5j} M_5 g + 2\beta_1 - \beta_1$

$= \alpha_{1j} M_1 g + \alpha_{3j} M_3 g + \alpha_{5j} M_5 g + \beta_1$

Hence there exist nontrivial values. So the lemma is proved. ∎

Corollary 2 *There exist non trivial values α_{1j}, α_{3j}, α_{5j}, ... $\alpha_{(2N+1)j}$, α_{1k}, α_{3k}, α_{5k}, ..., $\alpha_{(2N+1)k}$ such that*

$$\alpha_{1j} M_1 g + \alpha_{3j} M_3 g + \alpha_{5j} M_5 g + \ldots + \alpha_{(2N+1)j} + a_j \beta_1 =$$

$$\alpha_{1k}M_1g + \alpha_{3k}M_3g + \alpha_{5k}M_5g + \ldots + \alpha_{(2N+1)k} + a_k\beta_1$$

for some j and k. □

Recurrence Relation

To keep the memory requirements low we use Floyd's cycle detection algorithm. Checking i and $2i$ in a sequence is sufficient to find collision as proved in Proposition 11.1 in [27]. So we define a recurrence relation.

Let $r_1(0)$, $r_3(0)$, $r_5(0)$ and $r'(0)$ be small chosen integers (not all same). Let $E_0 = r_1(0)M_1g + r_3(0)M_3g + r_5(0)M_5g - r'(0)\beta_1$
Let $t_i = x(E_i)$ and bits$(\alpha_1) = b(\alpha_1)$

$$r_1(i) = \begin{cases} r_1(i-1)+1 & \text{if } t_i < l(\alpha_1) \times p/(l_u) \\ r_1(i-1) & \text{Otherwise} \end{cases}$$

$$r_3(i) = \begin{cases} r_3(i-1)+1 & \text{if } b(\alpha_1) \times p/(l_u+1) < t_i \leq [b(\alpha_1)+b(\alpha_3)] \times p/(l_u) \\ r_3(i-1) & \text{Otherwise} \end{cases}$$

$$r_5(i) = \begin{cases} r_5(i-1)+1 & \text{if } [b(\alpha_1)+b(\alpha_3)] \times p/(l_u) < t_i \text{ and} \\ & t_i \leq [b(\alpha_1)+b(\alpha_3)+l(\alpha_5)] \times p/(l_u) \\ r_5 & \text{Otherwise} \end{cases}$$

$$r'(i) = \begin{cases} r'(i-1)+1 & \text{if } t_i \equiv 0 \mod 3 \\ r'(i-1) & \text{Otherwise} \end{cases}$$

$E_i = r_1(t_i)M_1g + r_3(t_i)M_3g + r_5(t_i)M_5g - r'(t_i)\beta_1$
if E_i equals E_{2i} for some value of i then we get a relation which involves g and β_1 (which can be written in terms of β) As we know the order of g we can simplify for α
$r_1(t_i)M_1g + r_3(t_i)M_3g + r_5(t_i)M_5g - r'(t_i)\beta_1$

$= r_1(t_{2i})M_1g + r_3(t_{2i})M_3g + r_5(t_{2i})M_5g - r'(t_{2i})\beta_1 \bmod Ord(g)$
$(r_1(t_i)M_1 + r_3(t_i)M_3 + r_5(t_i)M_5 - r_1(t_{2i})M_1 - r_3(t_{2i})M_3 - r_5(t_{2i})M_5)\, g$

$= (r'(t_1) - r'(t_{2i}))\beta_1 \bmod Ord(g)$
Let $k_1 = (r_1(t_i)M_1 + r_3(t_i)M_3 + r_5(t_i)M_5 - r_1(t_{2i})M_1 - r_3(t_{2i})M_3 - r_5(t_{2i})M_5)$
Substituting the value of β_1
$k_1g = (r'(t_i) - r'(t_{2i}))(\beta - \alpha_2M_2g - \alpha_4M_4g) \bmod Ord(g)$ Rearranging gives

$k_1 g + (\alpha_2 M_2 + \alpha_4 M_4)(r'(t_i) - r'(t_{2i}))g = (r'(t_i) - r'(t_{2i}))\beta \bmod Ord(g)$
Let $k_2 = k_1 + (\alpha_2 M_2 + \alpha_4 M_4)(r_6(t_i) - r_6(t_{2i}))$ and
Let $k_3 = r'(t_i) - r'(t_{2i})$ then the equation becomes
$k_2 g = k_3 \beta \bmod Ord(g)$
So $k_2 k_3^{-1} g = \beta \bmod Ord(g)$ if k_3 is invertible. $k_3^{-1} k_2$ is the required α
If k_3 is not invertible we may repeat the method or try for values of α by substitution. Same type of calculation can be done when E_i equals $-E_{2i}$

Let the number of unknown bits be l_u. The possible number of values the above equation can take is 2^{l_u}. We will get a match in $O(\sqrt{2^{l_u}})$ (as in Pollard ρ method). So the time complexity is $O(\sqrt{2^l})$ in the best case. In the worst case it may go upto $O(\sqrt{p})$

5.4 Three parts of the key are revealed

If we know three parts of the key then we can write

$$\alpha = \alpha_1 M_1 + \alpha_2 M_2 + \alpha_3 M_3 + \alpha_4 M_4 + \alpha_5 M_5 + \alpha_6 M_6 + \alpha_7 M_7$$

we know α_2, α_4, α_6 substituting in the equation $\alpha g = \beta$ we get $\alpha_1 M_1 g + \alpha_3 M_3 g + \alpha_5 M_5 g + \alpha_7 M_7 g = \beta_1$ where $\beta_1 = \beta - \alpha_2 M_2 g - \alpha_4 M_4 g - \alpha_6 M_6$ Substituting the random values for α_2, α_4, α_6, we can solve for α similar to the case when two portions of the key are revealed. Similarly we can solve when n portions of the key are known.

5.5 Proposed Algorithm

The generator g and the key are related as $\alpha g = \beta$
We assume that n contiguous bits of the key are known.
The key (α) can be divided as follows
$\alpha = \alpha_1 M_1 + \alpha_2 M_2 + \cdots + \alpha_{2n+1} M_{2n+1}$.
So α_2, α_4, ..., α_{2n} are known.

Input: n known contiguous bits of the key i.e $\alpha[2]$, $\alpha[4]$, ..., $\alpha[2n]$ along with the position where they are occurring in the key.
g generator of the elliptic curve group
β public parameter

Output: α

begin

α can be written as

$\alpha = \alpha[1]M[1] + \alpha[2]M[2] + \cdots + \alpha[2n+1]M[2n+1]$;

`The values of M[ ] are adjusted as we know the positions of` $\alpha[]$;

$\beta_1 = \beta$;

for $i = 1$ *to* n **do**

$\beta_1 = \beta_1 - \alpha[2 \times i] \times M[2 \times i] \times g$

end

$P_1 = P_2 = \mathcal{O}$;

Array r is initialized to zero;

Array s is copied with the values of array r;

while *true* **do**

P_1= next$(r, M, g, \beta_1, r_{old}, P_1)$;

P_2=next $(s, M, g, \beta_1, s_{old}, P_2)$;

P_2=next $(s, M, g, \beta_1, s_{old}, P_2)$;

if $P_1 = P_2$ *or* $P_1 = -P_2$ **then**

$r_{old}[1]M[1]g + r_{old}[3]M[3]g + \cdots + r_{old}[2n+1]M[2n+1]g + \beta_1 r_{old}[2n+2]M[2n+2]g = (\pm)s_{old}[1]M[1]g + s_{old}[3]M[3]g + \cdots + s_{old}[2n+1]M[2n+1]g + \beta_1 s_{old}[2n+2]M[2n+2]g$;

`we got an equation involving` g `and` β `(as` β_1 `can be expressed in` β`) so we solve to get` α ;

return α

end

end

end

Algorithm 10: To find α when n portions (partial knowledge) of the key are known

begin PROCEDURE **next**

Input: Integer array t,
Integer array M,
generator g,
β_1, Integer array t_{old},
$point$ on the curve

Output: $Point$

copy array t to t_{old};

$frac = p$/(total unknown bits) ;

$temp = \mathcal{O}$;

$count$=number of contiguous unknown bits ;

for $i = 0$ *to* $count$ **do**

$temp = temp + t[i] \times M[i] \times g$

end

$temp = temp - t[count + 1] \times \beta_1$;

x=x-coordinate of temp;

un=0; flag=0;

for i*=0 to* $count$ **do**

un=un+i^{th} unknown bit length ;

if $x \leq un \times frac$ **then**

t[i]=t[i]+1;

flag=1;

break the for loop;

end

end

if *flag==0* **then**

$t[count + 1] = t[count + 1] + 1$

end

end

$point = temp$;

return $point$

Algorithm 11: Procedure next

5.6 Usage of the algorithm

We have given an algorithm where we know some parts of the key. We can apply the algorithm when the Least Significant Bits (LSB) and the Most Significant Bits(MSB) are unknown. We will now explain how to use the algorithm when known parts of the key contains LSB or MSB or both are known.

5.6.1 Some bits in the LSB are known

We know some parts of the key which include the Least Significant Bit (LSB) apart from some other parts of the key. So the key will look like (We use || symbol for concatenation)
α=unknown bits || known bits || unknown bits || known bits ||.....||unknown bits || know bits.
α= (known and unknown bits scattered) || known bits.
We know $\alpha \times g = \beta$. Substituting the value of α we get
((known and unknown scattered bits)$\times 2^{\text{No: of LSB bits known}}$ + (Known bits)) $\times g = \beta$
(known and unknown scattered bits)$\times (g \times 2^{\text{No: of LSB bits known}})$=$\beta$−(Known bits) $\times g$
(known and unknown scattered bits)$\times g'$=β'
where $g' = g \times 2^{\text{No: of LSB bits known}}$ and $\beta' = \beta -$ Known bits $\times\ g$
Now we apply the above proposed algorithm and get the unknown scattered bits.

5.6.2 Some bits in the MSB are known

We know some parts of the key which include the Most Significant Bit (MSB) apart from some other parts of the key So the key will look like
α=known bits || unknown bits || known bits ||.....||unknown bits || know bits || unknown bits.
α=known bits || (known and unknown bits scattered)
We know $\alpha \times g = \beta$.
Substuting the value of α we get
(known bits)$\times 2^l$+(know and unknown scattered bits) $\times g$=β
where l is number of bits in the block (known and unknown scattered bits)
Let β' = β - (known bits)$\times 2^l$. So now we have the equation as
(known and unknown scattered bits) $\times g$= β'.
Now we apply the above proposed algorithm and get the unknown scattered bits.

5.6.3 Some bits in both LSB and MSB are known

We know some parts of the key which include the Least Significant Bit (LSB) and we know some parts of the key which include the Most Significant Bit (MSB) apart from some other parts of the key. We first apply the trick described in the above Subsection 5.6.1 and then apply the trick described in the above Subsection 5.6.2. Now we apply the above proposed algorithm and get the key bits.

5.7 Analysis

Two parts of the key are revealed

Let us consider the Equation 5.1

$$\alpha_{1i}M_1g + \alpha_{3i}M_3g + \alpha_{5i}M_5g - a_i\beta_1 = R_i$$

From Lemma 2 we know that there exist atleast two distinct sets of values α_{1i}, α_{3i}, α_{5i}, a_i for each R_i. Similarly there exist atleast two distinct sets of values α_{1i}, α_{3i}, α_{5i}, a_i for each $-R_i$. We know M_1, M_3, M_5, g, β_1 are constants. Our aim is to find an equation which involves only g and β_1 along with the constants.

So we keep on substituting different values for α_{1i}, α_{3i}, α_{5i}, a_i and keep storing the value (α_{1i}, α_{3i}, α_{5i}, a_i ,R_i). If we get two coordinates of this form (α_{1j}, α_{3j}, α_{5j}, a_j, R_j) , (α_{1k}, α_{3k}, α_{5k}, a_k, R_j) we can eliminate R_j from the equation

$$\alpha_{1j}M_1g + \alpha_{3j}M_3g + \alpha_{5j}M_5g - a_j\beta_1 = R_j = \alpha_{1k}M_1g + \alpha_{3k}M_3g + \alpha_{5k}M_5g - a_k\beta_1$$

$$\begin{aligned}(\alpha_{1j}M_1 - \alpha_{1k}M_1 + \alpha_{3j}M_3 - \quad & \alpha_{3k}M_3 + \alpha_{5j}M_5 - \alpha_{5k}M_5)g = (a_j - a_k)\beta_1 \\ = \quad & (a_j - a_k)(\beta - \alpha_2M_2g - \alpha_4M_4g) \\ = \quad & (a_j - a_k)\beta - (a_j - a_k)(\alpha_2M_2 - \alpha_4M_4)g\end{aligned}$$

Let $f_1 = \alpha_{1j}M_1 - \alpha_{1k}M_1 + \alpha_{3j}M_3 - \alpha_{3k}M_3 + \alpha_{5j}M_5 - \alpha_{5k}M_5$, $f_2 = (a_j - a_k)$, $f_3 = (a_j - a_k)(\alpha_2M_2 - \alpha_4M_4)$. So the equation becomes

$$\begin{aligned}f_1g &= f_2\beta - f_3g \\ (f_1 + f_3)f_2^{-1}g &= \beta\end{aligned}$$

So the required α is $f_2^{-1}(f_1 + f_3)$.

Suppose we get the coordinates of the form
(α_{1j}, α_{3j}, α_{5j}, a_j, R_j) , (α_{1k}, α_{3k}, α_{5k}, a_k, $-R_j$). We can eliminate R_j from the equation

$$\alpha_{1j}M_1g + \alpha_{3j}M_3g + \alpha_{5j}M_5g - a_j\beta_1 = R_j = -(\alpha_{1k}M_1g + \alpha_{3k}M_3g + \alpha_{5k}M_5g - a_k\beta_1)$$

proceeding similarly as shown above we can get the value of α.

The values α_{1i} can take is 0 to $2^{\text{length}(\alpha_1)}$, α_{3i} can take is 0 to $2^{\text{length}(\alpha_3)}$, α_{5i} can take is 0 to $2^{\text{length}(\alpha_5)}$.

The total number of values (α_{1j}, α_{3j}, α_{5j}, a_j, R_j) can take is $2^{\text{length}(\alpha_1)} \times 2^{\text{length}(\alpha_3)} \times 2^{\text{length}(\alpha_5)} = 2^{\text{length}(\alpha_1)+\text{length}(\alpha_3)+\text{length}(\alpha_5)} = 2^{\text{no: unknown bits}}$

Let an urn contains n distinct numbered balls. We select the balls at random with replacement and make a list of the ball numbers we have selected. By Birthday problem and theory explained in Theorem 1 in Page 13, if we pick $\sqrt{n}$ balls there is 50% chance that we get a collision (some ball number in the list repeats).

Using Birthday problem if we generate $\sqrt{2^{\text{no: unknown bits}}}$ number of R_i there is high chance that there is collision. So the time complexity is $O(\sqrt{2^{(\text{unknown bits})}})$. Similarly if we know n portions of the key the time complexity is $O(\sqrt{2^{(\text{unknown bits})}})$

Our method works when n portions of the key are revealed whereas the earlier method [16] is applicable only when one portion of the key is revealed. The time complexity of our method is $O(\sqrt{2^{(\text{unknown bits})}})$ whereas the time complexity of [16] is $O(\sqrt{2p}/N^{\frac{1}{4}})$ where $N = 2^{\text{known bits}}$ and p can be written as $2^{\text{known + unknown bits}}$ so

$$\begin{aligned} O(\sqrt{2p}/N^{\frac{1}{4}}) &= O\left(\sqrt{2} \times 2^{\frac{\text{known + unknown bits}}{2}} / 2^{\frac{\text{known bits}}{4}}\right) \\ &= O\left(\sqrt{2} \times 2^{\frac{\text{known+unknown}}{2} - \frac{\text{known bits}}{4}}\right) \\ &= O\left(\sqrt{2} \times 2^{\frac{\text{known bits}}{4}} \times 2^{\frac{\text{unknown bits}}{2}}\right) \\ &= O\left(\sqrt{2} \times 2^{\frac{\text{known bits}}{4}} \times \sqrt{2^{\text{unknown bits}}}\right) \end{aligned}$$

So the time complexity of our method is $O(\sqrt{2^{(\text{unknown bits})}})$ and the time complexity of Gopalakrishnan et al. [16] is $O\left(\sqrt{2} \times 2^{\frac{\text{known bits}}{4}} \times \sqrt{2^{\text{unknown bits}}}\right)$

5.7.1 Implementation

We have derived the theoretical time complexity above. Now we are looking at an empirical analysis of time complexities. We have implemented Gopalakrishnan et al. algorithm and our algorithm using C and GMP [14] library.

C1, C2, .. are the names chosen for the curves. prime column denotes the prime number chosen for the prime field Two coefficients a, $b \in F_p$ that define the equation of the elliptic curve (i.e $y^2 = x^3 + a\,x + b$) base point is the generator g of the curve key is chosen (and it

is alpha). known bits are the bits given as the input to the program along with the position information.

Table 5.1: Instances of Discrete Log Problems

Curve	prime	a	b	base point	order of base point	key(in binary)	beta= key*base point
C1	104677	10	28	(59010, 27440)	52552	1010 0001 0111 01 (14 bits)	(78038, 56158)
C2	104683	23	84	(28451, 182)	104326	1010 0001 0111 01(14 bits)	(75645, 40063)
C3	1299 919	37	837	(857959, 555867)	1299556	1100 0101 1110 1010 1 (17bits)	(972257, 858774)
C4	1548 5933	76	689	(15038473, 6300452)	7739768	1111 0100 1111 1000 0111 (20 bits)	(4036478, 979519)

Table 5.2: Performance of Gopalakrishnan et al. algorithm and the proposed algorithm

Name of the curve	known bits	no: of iterations goplakrishna et al algorithm	No: of iterations in our algorithm
C1	01 011	1555	660
C2	0111	3341	522
C3	10101	2411	1312
C4	11000	28147	5610

Table 5.3: Instances of Discrete Log problems of bit length more than 30

Curve	prime	a	b	base point (g)	order of base point	key(in binary)	beta=key*base point
C6	10000 00001 9 (34 bits)	10	30	(73854 4445, 78442 44386)	99999 84888	11000 10010 10111 00101 10110 10001 001	(74584 18165, 50567 30098)
C7	10000 00000 03 (37 bits)	50	40	(64896 057935, 71623 034177)	99999 64978 2	10010 11000 11000 01010 00011 10000 0101	(88841 696421, 16096 25674 2)
C8	10000 00000 039 (40 bits)	130	30	(64250 3508979, 63294 0462198)	74886 21289 81	10101 11001 01101 11010 10010 10000 11010 10101	(17506 6691199, 83166 13741 61)
C9	80013 11119 213 (43 bits)	410	352	(31386 45431772, 26843 2051400)	80013 10008 868	10001 00110 11000 00101 01000 01011 01100 11000 110	(48884 44800659, 79784 29728 332)

Table 5.4: Performance of our algorithm

Name of the curve	No of known parts of the key	known bits		Time taken	No: of iterations in our algorithm	$\sqrt{2^{\text{unknown bits}}}$
		Position	bits			
C6	1	6	10010	463 sec	484972	32768
C6	2	11 21	10111 10110	382 sec	309550	4096
C6	3	6 16 26	10010 00101 10001	80 sec	63243	1024
C7	1	11	11000	462 sec	466414	65536
C7	2	11 22	11000 00111	372 sec	300216	16384
C7	3	6 16 26	11000 01010 10000	297 sec	201186	2048
C8	1	30	01101	2015 sec	1128913	262144
C8	2	11 21	01101 10010	1599	1091760	32768
C8	3	6 16 26	11001 11010 10000	1500	935598	8192
C9	2	11 26	11000 01011	10487 sec	3570258	131072
C9	2	21 31	01000 01100	9977 sec	3472669	131072
C9	3	11 21 31	11000 01000 01100	2977 sec	1488814	16384
C9	4	5 15 25 35	10011 00010 00101 01100	1243 sec	627555	4096

5.8 Parallelised algorithm to solve DLP from partial knowledge of the key

The proposed algorithm can be modified to run on a cluster. In the serial version only one instance of the Algorithm 10 (Page No: 48) will be running. In the parallel version each node will be running a copy of Algorithm 10. So we have to initialise the array r with different values for different nodes. The difference between $r[0]$ of node i and $r[0]$ of node j should be high, so that both nodes does not generate the same sequence.

Example 4 Let the initial r array of node 1 is $r[]$=$\{0,0,0\}$. The next function (Procedure 11 on Page No: 49) takes care of which value of r array need to be incremented. The inputs to next function are $r, M, g, \beta_1, r_{old}, P_1$. Array M is position array which is fixed. g is generator which is fixed. β_1 is the value calculated before the iterations so it is fixed. Let us assume that, in the first iteration next function has increment $r[0]$. So the r array is $\{1, 0, 0\}$.

Similarly, let the initial r array of node 2 is $r[]$=$\{0,0,0\}$. The input values for the next function in node 2 is same as the node 1. So the r array after first iteration is $\{1, 0, 0\}$. So both node 1 and node 2 generate same r array in each iterations. This means node 1 and node 2 are following same path and gives raise to trivial collision. With trivial collisions we can not solve the ECDLP problem.

The distance (positive difference) between $r[0]$ of node 1 and $r[0]$ of node 2 is 0. The distance between $r[1]$ of node 1 and $r[1]$ of node 2 is 0. The distance between $r[2]$ of node 1 and $r[2]$ of node 2 is 0. The total distance between corresponding $r[i]$ of node 1 and $r[i]$ of node 2 is 0.

Example 5 Let the initial r array of node 1 is $r[]$=$\{0,0,0\}$. Let us assume that, in the first iteration next function has increment $r[0]$. So the r array is $\{1, 0, 0\}$. Let the initial r array of node 2 is $r[]$=$\{0,0,1\}$. In the first iteration there is equal probability of incrementing each $r[i]$. Let us assume that, in the first iteration next function has increment $r[1]$. So the r array is $\{1, 0, 1\}$. There is some chance that both node 1 and node 2 generate different set of r array. Means they follow different paths. There is a possibility of nontrivial collisions. The total distance between corresponding $r[i]$ of node 1 and $r[i]$ of node 2 is 1. Here the distance is higher than the previous example. □

The algorithm fixes some criteria (e.g more number of ones than zeros etc) and if the P_i matches that criteria they will be sent to server node. The server node checks for two non trivial matching points and calculates the key. The same Algorithm 2 is share by all the nodes without any modification.

The generator g and the key are related as $\alpha g = \beta$ We assume that n contiguous bits of the key are known.

The key (α) can be divided as follows

$\alpha = \alpha_1 M_1 + \alpha_2 M_2 + \cdots + \alpha_{2n+1} M_{2n+1}$.

So α_2, α_4, ..., α_{2n} are known.

Input: n known contiguous bits of the key i.e $\alpha[2]$, $\alpha[4]$, ..., $\alpha[2n]$ along with the position where they are occuring in the key.
g generator of the elliptic curve group
β public parameter

Output: α

begin

α can be written as

$\alpha = \alpha[1]M[1] + \alpha[2]M[2] + \cdots + \alpha[2n+1]M[2n+1]$;

`The values of M[ ] are adjusted as we know the position's of` $\alpha[]$;

$\beta_1 = \beta$;

for $i = 1$ *to* n **do**

$\beta_1 = \beta_1 - \alpha[2 \times i] \times M[2 \times i] \times g$

end

$P_1 = P_2 = \mathcal{O}$;

Array r is initialized to with random integers depending on the node number;

while *true* **do**

P_1= next($r, M, g, \beta_1, r_{old}, P_1$) ;

if P_1 *matches certain criteria* **then**

Send P_1, r, r_{old} to central / server node.

end

`Examples of the criteria are 1) No: of one's +10 > No: of zero's in` P_1, `2)` P_1 `has 5 consecutive zeros 3)` P_1 `has 5 consecutive ones or mixture of these`

end

end

Algorithm 12: Procedure on a node

The server node upon receiving the new point checks with the existing points and if it finds it to be non trivial match with the existing point we calculate the key as we have done in Algorithm 1.

$$r^i_{old}[1]M^i[1]g + r^i_{old}[3]M^i[3]g + \cdots + r^i_{old}[2n+1]M^i[2n+1]g + \beta_1 r^i_{old}[2n+2]M^i[2n+2]g = (\pm)s^j_{old}[1]M^j[1]g + s^j_{old}[3]M^j[3]g + \cdots + s^j_{old}[2n+1]M^j[2n+1]g + \beta_1 s_{old}[2n+2]M^j[2n+2]g$$

We got an equation involving g and β (as β_1 can be expressed in β)
so we solve to get α

We have discussed about our approach in solving the ECDLP from partial knowledge of the key. We have done the theoretical and empirical analysis of the proposed algorithm. Our method works when n portions of the key are revealed whereas the earlier method [16] is applicable only when one portion of the key is revealed. So the time complexity of our method is $O(\sqrt{2^{\text{(unknown bits)}}})$ and the time complexity of Gopalakrishnan et al. [16] is $O\left(\sqrt{2} \times 2^{\frac{\text{known bits}}{4}} \times \sqrt{2^{\text{unknown bits}}}\right)$. We have developed a parallel version of the proposed algorithm to solve discrete logarithm from partial knowledge of the key.

Chapter 6

Conclusion

The communication media rely on many cryptographic protocols for securing the data. Some of these cryptographic protocols in turn depend on the hard problems such as the Discrete Logarithm problem, factoring, Elliptic curve discrete log problem, etc for their security. In this thesis, the Cryptanalysis of Elliptic Curve Discrete Logarithm Problem (ECDLP) is considered. There is no algorithm to solve Elliptic curve discrete log problem in polynomial time on binary representation of the input.

The Elliptic Curve Digital Signature Algorithm (ECDSA), is the most widely standardized elliptic curve-based signature scheme, appearing in the ANSI X9.62, FIPS 186-2, IEEE 1363-2000 and ISO/IEC 15946-2 standards as well as several draft standards [3, 12]. We have proposed the two key signature scheme. The security features are analysed in detail. We have proposed an attack on the proposed signature scheme. As Two key ECDSA is backward compatible, subset of domain parameters of Two key ECDSA are required to use One key ECDSA. The user can use either one key or two key ECDSA algorithms depending on the security requirements for signing the documents. We have demonstrated the application of Two key ECDSA in Digital Certificates and counter signing.

We have found error in Gopala Krishnan et al. [16] algorithm and suggested necessary modification to it. We generalised the algorithm to cases when any arbitrary regions or even bits of the key are known and came up with the new algorithm discussed in Chapter 5. The new algorithm is also shown to be more efficient when compared with the existing algorithm.

Finally, the different algorithms proposed in the thesis have been implemented in GMP and Sage libraries and the results show the effectiveness and correctness of the theoretical computational complexities.

Bibliography

[1] List of unsolved problems in computer science — Wikipedia, The Free Encyclopedia, 2015. [Online accessed 6-November-2015].

[2] GCHQ trio recognised for key to secure shopping online. `http://www.bbc.com/news/uk-england-gloucestershire-11475101`, 5 Oct 2010.

[3] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid. Recommendation for Key Management–Part 1: General. *NIST Special Publication*, pages 800 – 857, 2005.

[4] Joppe W Bos, J Alex Halderman, Nadia Heninger, Jonathan Moore, Michael Naehrig, and Eric Wustrow. Elliptic curve cryptography in practice. *Microsoft Research. November*, 2013.

[5] R. Canetti, J.B. Friedlander, S.Konyagin, M.Larsen, D. Lieman, and I. Shparlinski. On statistical properties of distribution of diffie– hellman distribution. *Israel J. Math*, 120:23–46, 2000.

[6] J. Cannon et al. The MAGMA computational algebra system. *Software available on line* `http://magma.maths.usyd.edu.au`, 2005.

[7] Dave Cooper. Internet X. 509 public key infrastructure certificate and certificate revocation list (CRL) profile. 2008.

[8] Darrel Hankerson, Affred Menezes, and Scott Vanstone. *Guide to Elliptic Curve Cryptography*. Springer Verlag, 2004.

[9] Whitfield Diffie. The first ten years of public-key cryptography. *Proceedings of the IEEE*, 76(5):560–577, 1988.

[10] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[11] Andreas Enge. *Elliptic Curves and Their Applications to Cryptography - An Introduction*. Springer, 1999.

[12] Standards for Efficient Cryptography Group (SECG). Sec 1: Elliptic curve cryptography, September 2000. http://www.secg.org/download/aid-385/sec1_final.pdf.

[13] Martin Gardner and WW Ball. Mathematical games. *Scientific American*, 231:187–191, 1974.

[14] Torbjörn et al. Granlund. Gmp, the gnu multiple precision arithmetic library.

[15] Jeffrey Hoffstein, Jill Catherine Pipher, Joseph H Silverman, and Joseph H Silverman. *An introduction to mathematical cryptography*. Springer, 2008.

[16] K.Gopalakrishnan, Nicolas Theriault, and Chui Zhi Yao. Solving discrete logarithms from partial knowledge of the key. In K.Srinathan, C.Pandu Rangan, and M.Yung, editors, *Indocrypt 2007*, LNCS 4859, pages 224–237. Springer-Verlag Berlin, 2007.

[17] S. Lang. *Algebra*. Graduate Texts in Mathematics. Springer-Verlag, New York, 2002.

[18] Laurie Law, Alfred Menezes, Minghua Qu, Jerry Solinas, and Scott Vanstone. An efficient protocol for authenticated key agreement. *Designs, Codes and Cryptography*, 28(2):119–134, 2003.

[19] A. Menezes, S. Vanstone, and T. Okamoto. Reducing elliptic curve logarithms to logarithms in a finite field. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, page 89. ACM, 1991.

[20] V.S. Miller. Use of elliptic curves in cryptography. *Lecture notes in computer sciences; 218 on Advances in cryptology—CRYPTO 85*, pages 417–426, 1986.

[21] N.Anil Kumar and Chakravarthy Bhagvati. Two key signature scheme with application to digital certificates. In *Recent Advances in Information Technology (RAIT)*, pages 19–22, ISM, Dhanbad, 2012.

[22] N.Anil Kumar and Chakravarthy Bhagvati. Efficient algorithm to solve dlp from partial knowledge of the key. In *Proceedings of International Conference on Computer Communication & Informatics*. IEEE, 2013.

[23] N.Anil Kumar, R.Tandon, and Chakravarthy Bhagvati. Modified Elliptic Curve Digital Signature Algorithm. In R.Bharati and Indra Rajasingh, editors, *Proceedings of the International Conference on Mathematics and Computer Science 2009*, volume 1, pages 304–306, 2009.

[24] VK Pachghare. *Cryptography and information security*. PHI Learning Pvt. Ltd., 2008.

[25] Stephen C Pohlig and Martin E Hellman. An improved algorithm for computing logarithms over gf (p) and its cryptographic significance. 1978 january: Ieee transactions on information theory. *Article submitted on*, 1976.

[26] John M Pollard. Kangaroos, monopoly and discrete logarithms. *Journal of cryptology*, 13(4):437–447, 2000.

[27] E. Prisner. *Graph Dynamics*. Pitman Research Notes In Mathematics Series. Pitman Research Notes In Mathematics Series, Longman House, Burnt Mill, Harlow, 1995.

[28] Victor Shoup. A proposal for an iso standard for public key encryption (version 2.1). *IACR E-Print Archive*, 112, 2001.

[29] WA Stein et al. Sage Mathematics Software (Version 6.2), 2009. `http://www.sagemath.org/`.

[30] L.C. Washington. *Elliptic Curves: Number Theory and Cryptography*. CRC Press, 2003.

[31] Wikipedia. James H. Ellis — Wikipedia, The Free Encyclopedia, 2015. [Online; accessed 28-November-2015].

Appendix A

Implementation

In this appendix we will provide the sample codes we have developed.

A.1 Sage sample code

Sample implementation of the proposed Two key ECDLP in Sage

```
p=252151;
a=337;
b=47;
#hash of the message
e=28;
f=GF(p,'x');
E=EllipticCurve(f,[a,b]);
order=E.cardinality();
print E.gens();
P=E(184870,17195);
P1=E(129882,52865);
d=20;
d1=41;
print "The private keys d=",d," d1=",d1;
R=d*P+d1*P1;
Q=d*P;
print "P=",P,"P1=",P1,"R=",R;
k1=5;
```

```
k2=8;
print "The random numbers selected are k1=",k1," k2=",k2;
S=k1*P+k2*P1;
print "S=",S;
print "Hash of the message is=",e;
x1=S[0];
x=Integer(x1);
s1=e*k1+x*d;
s2=e*k2+x*d1;
s1=Integer(mod(s1,order));
s2=Integer(mod(s2,order));
print "Signature is", x1,s1,s2;
print "--------------------signature generated----------"
u1=s1*P+s2*P1;
print "u1=",u1;
T=E.lift_x(x1,all=True)[0];
T1=E.lift_x(x1,all=True)[1];
print "T=",T,"T1=",T1;
u2=e*T+x*R;
print "u2=",u2;
if u1==u2:
    print "Accept the    signature";
else:
    u2=e*T1+x*R;
    print "u2=",u2;
    if u1==u2:
        print "Accept the signature";
    else:
        print "Reject the signature";
-------------------------------------------
[(45699 : 72850 : 1), (167119 : 148912 : 1)]
The private keys d= 20  d1= 41
P= (184870 : 17195 : 1) P1= (129882 : 52865 : 1) R= (69369 : 149658 : 1
The random numbers selected are k1= 5  k2= 8
S= (237658 : 101436 : 1)
Hash of the message is= 28
Signature is 237658 216328 166150
--------------------signature generated----------
u1= (154209 : 95062 : 1)
```

```
T= (237658 : 101436 : 1) T1= (237658 : 150715 : 1)
u2= (154209 : 95062 : 1)
Accept the    signature
-----------------------------------------------------
```

A.2 Function implemented

In this section we give a brief description of the functions we have developed for the implementation of the existing Gopalakrishnan et al. and our proposed schemes.

We have defined a class BigInt which overloads the operators like +, -, *, /, %, =, etc. Initialisation of memory and freeing of memory is automatically taken care of in BigInt by constructors and destructors. The class BigInt uses gmp integers so it supports arbitrary number of digits.

```
BigInt a,b,c;
  a=10;
  b=20;
  c=a*b;
```

We have defined a class EC which provides the operations on elliptic curves. EC is built using BigInt so it supports arbitrary number of digits. The operator + is over loaded so we can use it add the points on the elliptic curve. The operator * is overloaded so we can use it for scalar multiplication.

`EC::setCurve(a,b,p)` This static function uses `a`, `b` and `p` to define elliptic curve $y^2 = x^3 + \mathtt{a}\, x + \mathtt{b} \bmod P$

`EC(BigInt x1, BigInt y1)` We can use this constructor to create point on the elliptic curve or use `set` method if we wish to create the point later.

`EC.set(BigInt x1, BigInt y1)` We can use this method to define the point after we create the object.

```
EC::setCurve(132,46,137);
EC p1(2,27), p2, p3, p4;
p2.set(5,134);
```

```
p3=p1+p2;
p4=2*p1;
```

`BigInt lam_ell(EC g, EC beta, BigInt a, BigInt b, BigInt order)`
The function `lam_ell` implements pollard's lambda algorithm. `g` is the generator point on the elliptic curve. `beta` is another point on the elliptic. Order of the `g` is `order`. The function returns `alpha` where `alpha` satisfies `alpha` $\times$ `g` = `beta` and `a` $\leq$ `alpha` $\leq$ `b`