HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

MASTER THESIS

Mr.
Saumik Barua

Algorithms For Discrete Logarithms

2022

Faculty of **Applied Computer Sciences and Biosciences**

MASTER THESIS

Algorithms For Discrete Logarithms

Author:
Saumik Barua

Study Programme:
Applied Mathematics For Network and Data Sciences

Seminar Group:
MA18w1-M

First Referee:
Prof. Dr. Klaus Dohmen

Second Referee:
Prof. Dr. Peter Tittmann

Mittweida, February 2022

Acknowledgement

I would like to express my gratitude and special thanks to my supervisor, Prof. Dr. Klaus Dohmen, for his guidance and support throughout this entire process. I also would like to extend my special thanks to Prof. Dr. Peter Tittmann for his valuable advice at different times throughout the program.

I wish to show my deep appreciation to my friends and well-wishers for their continuous support in many ways. Finally, I would like to express my sincere gratitude and thanks to my parents for providing tireless support and inspiration.

Bibliographic Information

Barua, Saumik: Algorithms For Discrete Logarithms, 89 pages, 5 figures, Hochschule Mittweida, University of Applied Sciences, Faculty of Applied Computer Sciences and Biosciences

Master Thesis, 2022

Abstract

Due to the intractability of the Discrete Logarithm Problem (DLP), it has been widely used in the field of cryptography and the security of several cryptosystems is based on the hardness of computation of DLP. In this paper, we start with the topics on Number Theory and Abstract Algebra as it will enable one to study the nature of discrete logarithms in a comprehensive way, and then, we concentrate on the application and computation of discrete logarithms. Application of discrete logarithms such as Diffie Hellman key exchange, ElGamal signature scheme, and several attacks over the DLP such as Baby-step Giant-step method, Silver Pohlig Hellman algorithm, etc have been analyzed. We also focus on the elliptic curve along with the discrete logarithm over the elliptic curve. Attacks for the elliptic curve discrete logarithm problem, ECDLP have been discussed. Moreover, the extension of several discrete logarithms-based protocols over the elliptic curve such as the elliptic curve digital signature algorithm, ECDSA have been discussed also.

I. Contents

Contents I
List of Figures II
List of Tables III
1 Introduction 1
2 Preliminaries on Number Theory and Abstract Algebra 3
2.1 Topics on Number Theory 3
2.2 Topics on Abstract Algebra 13
3 Discrete Logarithms: Application and Attack 19
3.1 Discrete Logarithms 19
3.2 Discrete Logarithm Based Cryptography 20
3.2.1 Diffie Hellman Key Exchange 20
3.2.2 ElGamal Cryptosystem 22
3.2.3 ElGamal Signature Scheme 23
3.3 Algorithms For Discrete Logarithm Problem 29
3.3.1 Baby-Step Giant-Step Algorithm 29
3.3.2 Index Calculus Method 31
3.3.3 Pollard Kangaroo Method 34
3.3.4 Silver Pohlig Hellman Algorithm 36
4 Elliptic Curve, Elliptic Curve Discrete Logarithm: Application and Attack 45
4.1 Topics on Elliptic Curve 45
4.2 Elliptic Curve Discrete Logarithm Problem 53
4.3 Elliptic Curve Discrete Logarithm Based Cryptography 54
4.3.1 Diffie Hellman Key Exchange over Elliptic Curve 54
4.3.2 Elliptic Curve ElGamal Cryptosystem 55
4.3.3 Elliptic Curve Digital Signature Algorithm 57
4.4 Classical Algorithms for Elliptic Curve Discrete Logarithms 59
4.4.1 Baby-Step Giant-Step Method for ECDLP: 59
4.4.2 Silver Pohlig Hellman Attack for ECDLP 60

5 Conclusion 65

A Sage Code for Applications and Attack on Discrete Logarithms 67

Bibliography 87

II. List of Figures

3.1 Figure for $3^x \pmod{79}$ 19

4.1 Figure for $y^2 = x^3 - x$ over $\mathbb{R}$ 46

4.2 Figure for $y^2 = x^3 + 6x + 16$ over $\mathbb{R}$ 46

4.3 Addition of two points on Elliptic curve: 48

4.4 Figure for $y^2 \equiv x^3 + 6x + 16 \pmod{1259}$: 52

III. List of Tables

2.1 Addition in $\mathbb{Z}_7$ 7

2.2 Multiplication in $\mathbb{Z}_7$ 7

2.3 Orders of elements of $\mathbb{Z}_{11}^*$ 15

2.4 Subgroups of $\mathbb{Z}_{11}^*$ 16

4.1 Finding points on $E : y^2 = x^3 + 6x + 4$ over the field F_{11}. 51

1 Introduction

Logarithms, invented by Scottish Mathematician John Napier, can be termed as the inverse of exponentiation operation. x is the logarithm of y to the base g denoted by $x = log_g y$, if $y = g^x$ where $g,x,y \in \mathbb{R}$. Let g,y are given, then, finding the value of x is termed as the logarithm problem. This problem seems simple since $log_g y = \frac{ln\ y}{ln\ g}$. Nevertheless, the case is not same for Discrete Logarithm Problem, DLP. Mainly, in terms of difficulty level of computation, there are three types of DLP: DLP in $\mathbb{Z}_n$, DLP in $\mathbb{Z}_n^*$, DLP in $E(\mathbb{F}_p)$ [25]. Computing DLP in additive group $(\mathbb{Z}_n, +)$ is easy but computing DLP in multiplicative group $(\mathbb{Z}_n^*, \cdot)$ and elliptic curve $(E(\mathbb{F}_p))$ is hard. Whitfeld Diffe and Martin Hellman proposed the Diffie Hellman key exchange as the initialization of public key cryptography in 1976. ElGamal cryptosystem and ElGamal signature scheme (based on DLP) were published by ElGamal in 1985. From then till today, there have been significant studies and advances for discrete logarithms. There are no efficient algorithms to solve DLP for classical computers as of today.

For the motivation, let us assume that Alice and Bob want to communicate securely whereas Trudy, an eavesdropper who wants to spy on the communication between Alice and Bob. Therefore, Alice and Bob would like to ensure confidentiality, authenticity, integrity, and non-repudiation. Digital signatures come up with the properties of authentication, integrity, and non-repudiation [3]. In the real world, a lot of digital signatures are being created and verified in every second by different institutions such as financial organizations or banks. Therefore, the demand for public-key digital signatures is increasing.The security of digital signatures and many cryptographic schemes depend on the intractability of discrete logarithm problems. This paper has been structured as follows:

In Chapter 2, several important concepts on number theory and abstract algebra including definitions, examples, and theorems have been discussed.

In Chapter 3, we discuss about discrete logarithms and several computation techniques of solving DLP. We have also concentrated on the discrete logarithm based cryptographic systems.

In Chapter 4, elliptic curves and elliptic curve discrete logarithm problem, ECDLP have been studied. Moreover, algorithms for ECDLP and extension of DLP in the elliptic curve analogue have been discussed.

In the appendix, the codes for different schemes and algorithms implemented in SageMath have been provided.

2 Preliminaries on Number Theory and Abstract Algebra

2.1 Topics on Number Theory

Definition 2.1 (Divisibility) Let a and b be two integers where $a \neq 0$. Then, we say that a *divides* b (or a is *divisor* of b) if there is an integer c such that $b = ac$. This is denoted by $a \mid b$.

A few basic properties of divisibility are given below [14].

Proposition 2.2 (Properties of divisibility) *For all $a, b, c \in \mathbb{Z}$, the following statements hold:*

1. $a \mid 0$ *and* $a \mid a$
2. *If* $a \mid b$ *and* $b \mid c$, *then* $a \mid c$
3. *If* $a \mid b$ *and* $b \mid a$, *then* $a = \pm b$
4. *If* $a \mid b$ *and* $a \mid c$, *then* $a \mid (bm + cn)$ *for any* $m, n \in \mathbb{Z}$

Proof:

1. Since $0 = a \cdot 0$, it can be considered that $c = 0$ in the definition and we get $a \mid 0$. Since $a = a \cdot 1$, we consider $c = 1$ which shows that $a \mid a$.

2. If $a \mid b$ and $b \mid c$, there exists m and n such that $b = am$ and $c = bn$. So, $c = (am) \cdot n = a(mn)$. Hence, $a \mid c$.

3. If $a \mid b$ and $b \mid a$, there are integers m and n such that $b = am$ and $a = bn$. So, $a = bn = (am) \cdot n = a(mn)$. Therefore, $mn = 1$. Since, m and n both are integers and $mn = 1$, either both are positive ($m = 1, n = 1$) or both are negative ($m = -1, n = -1$). Therefore, $a = \pm b$.

4. If $a \mid b$ and $a \mid c$, it can be written that there exist integer l_1 and l_2 such that $b = al_1$ and $c = al_2$. So, $bm + cn = al_1(m) + al_2(n)$ = $a(l_1 m + l_2 n)$. Hence, $a \mid (bm + cn)$.

□

Definition 2.3 The *greatest common divisor* (gcd) of two integers a and b is the largest positive integer that divides both a and b. It is denoted by (a,b) or gcd(a,b).

Definition 2.4 Two integers a and b are *coprime* or *relatively prime* if gcd$(a,b) = 1$.

Remark 2.5 Any two consecutive integers are always coprime.

2.1.1 Euclidean Algorithm and Extended Euclidean Algorithm

Euclidean algorithm is an standard method to compute greatest common divisor. The following described method has been adopted from [2].

Let a and b be two integers and $a > b$, if not we consider the greater number as a.

Euclidean Algorithm: EA(a,b)
Require: a,b integers
Ensure: $a > b$
  if $b == 0$ **then**
    return absolute(a)
  else
    return EA$(b, a \bmod b)$

Let us consider the coefficients and remainders at each step of the Euclidean algorithm on input (a,b). Let $a_0 = a$, $a_1 = b$ Then,

$$\begin{aligned} a_0 &= q_0 a_1 + a_2 \\ a_1 &= q_1 a_2 + a_3 \\ &\vdots \\ a_{n-2} &= q_{n-2} a_{n-1} + a_n \\ a_{n-1} &= q_{n-1} a_n \end{aligned}$$

where,

$$q_i = \left\lfloor \frac{a_i}{a_{i+1}} \right\rfloor \text{ for any } i \in \{0,1,\ldots,n-1\}.$$

Let the number of division steps of Euclidean algorithm be n and expressed as $E(a,b) = n$ where n is an integer. We have, $a_n = \gcd(a,b) = d$.

The Euclidean algorithm can be extended so that not only the greatest common divisor d of two integers a and b is obtained but also integers x and y satisfying $ax + by = d$.

The Extended Euclidean Algorithm: EEA(a,b)

Require: a,b integers
Ensure: $a > b$
  if $b == 0$ **then**
    return absolute(a), sign(a), 0
  end if
  $q \leftarrow a//b$
  $r \leftarrow a \mod b$
  $d, bb, rr \leftarrow$ recurse EEA(b, r)
  return $d, rr, bb - q * rr$ =0

2.1.1.1 Worst case analysis

Theorem 2.6 (Lame's theorem [2]) *The number of division steps in the Euclidean algorithm on input (a,b) is bounded above by 5 times the number of decimal digits in min(a,b).*

Lemma 2.7 ([2]) *Let a,b be integers where $a > b > 0$ and the Euclidean algorithm on input (a,b) performs n division steps. Then $b \geq F_{n+1}$.*

Proof: Mathematical induction can be used to prove the lemma.

For $n = 1$, $F_2 = 1$. Hence, $F_{n+1} \leq b$ holds.

For $n = 2$, $F_3 = 2$. Hence, $F_{n+1} \leq b$ holds.

For $n \geq 3$, following the algorithm, it can be shown that $F_{n+1} \leq b$ holds. □

If the number of division steps for Euclidean algorithm is n and the number of decimal digits in min(a,b) is q, then according to the Lame's theorem, $n \leq 5q$.

Therefore, an upper bound for n has been provided for $F_{n+1} \leq b$.

Lemma 2.8 ([24]) *Let a and b be two integers where gcd$(a,b) = d$. Then, there exists integers x and y such that $ax + by = d$ = gcd(a,b). In case of gcd$(a,b) = 1$, there exists x,y such that $ax + by = 1$.*

Remark 2.9 For finding the multiplicative inverse of a modulo b, extended euclidean algorithm can be applied to find integers x and y such that $ax + by = 1$ where gcd$(a,b) = 1$. Here, it should be noted that $x \equiv a^{-1} \pmod{b}$.

2.1.2 Congruence and modular arithmetic

Definition 2.10 (Congruence) Let a, b and n be integers, $n \neq 0$. Then, $a \equiv b \bmod n$ if $n \mid (a-b)$. In other words, a is congruent to b modulo n if n divides $a-b$. The integer n is said to be the modulus of the congruence.

Definition 2.11 A *congruence class* $[a]_n$ is the set of all integers that have the same remainder a when divided by n.

Example 2.12 In congruence modulo 5,

$$[3]_5 = \{3, 3 \pm 5, 3 \pm 10, 3 \pm 15, \dots\} = \{\dots, -12, -7, -2, 3, 8, 13, 18, \dots\}$$

A few properties of congruences are given below [14].

Proposition 2.13 (Properties of the congruences) *Let a, b, c, n be integers, $n \neq 0$. Then following properties hold:*

1. $a \equiv 0 \ (mod\ n)$ *iff* n *divides* a
2. $a \equiv a \ (mod\ n)$ *[Reflexivity]*
3. *If* $a \equiv b \ (mod\ n)$, *then* $b \equiv a \ (mod\ n)$ *[Symmetry]*
4. *If* $a \equiv b \ (mod\ n)$ *and* $b \equiv c \ (mod\ n)$, *then* $a \equiv c \ (mod\ n)$ *[transitivity]*

Proof:

1. $a \equiv 0 \pmod{n}$ implies $a - 0 = a$ is a multiple of n which means n divides a.
2. $a - a = 0 \cdot n$. Hence, $a \equiv a \pmod{n}$.
3. $a \equiv b \pmod{n}$ implies that $a - b \equiv 0 \pmod{n}$ which means $a - b$ is a multiple of n. So, $a - b = m \cdot n$ or $b - a = (-m)n$. Therefore, $b \equiv a \pmod{n}$. If we switch up a and b, the reverse implication will be obtained.
4. If $a \equiv b \pmod{n}$ then $a = b + mn$ and if $b \equiv c \pmod{n}$ then $b = c + nl$.
   So, $a = c + nl + mn = c + (l + m)n$. Hence, $a \equiv c \pmod{n}$.

□

Proposition 2.14 ([24]) *Let a, b, c, d, n be integers and $n \neq 0$. If $a \equiv b \ (mod\ n)$ and $c \equiv d \ (mod\ n)$, then $a + c \equiv b + d \ (mod\ n)$, $a - c \equiv b - d \ (mod\ n)$, $ac \equiv bd \ (mod\ n)$.*

Proof: We can write, $a = b + mn$ and $c = d + ln$ for integers l and m.

Then,

$$a+c=b+d+mn+ln=b+d+n(m+l).$$

So,

$$a+c\equiv b+d \pmod{n}.$$

Similarly, $a-c=b-d+mn-ln=b-d+n(m-l)$. So, $a-c\equiv b-d \pmod{n}$.
Now,

$$\begin{aligned} \text{ac} &= (b+nk)(d+nl) \\ &= bd+bnl+dnk+n^2kl \\ &= bd+n(bl+dk+nkl) \end{aligned}$$

Hence, $ac\equiv bd \pmod{n}$. □

Two tables of addition in $\mathbb{Z}_7$ and Multiplication in $\mathbb{Z}_7$ are given below:

+	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	2	3	4	5	6	0
2	2	3	4	5	6	0	1
3	3	4	5	6	0	1	2
4	4	5	6	0	1	2	3
5	5	6	0	1	2	3	4
6	6	0	1	2	3	4	5

Table 2.1: Addition in $\mathbb{Z}_7$

·	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6
2	0	2	4	6	1	3	5
3	0	3	6	2	5	1	4
4	0	4	1	5	2	6	3
5	0	5	3	1	6	4	2
6	0	6	5	4	3	2	1

Table 2.2: Multiplication in $\mathbb{Z}_7$

Theorem 2.15 (Chinese Remainder Theorem([7]) *Let $n_1, n_2 \ldots, n_k$ are pairwise relatively prime [i.e, $gcd(n_i, n_j) = 1$, if $i \neq j$] and $a_1, a_2, \ldots a_k$ are integers. Consider the following system of congruences:*

$$\begin{aligned} x &\equiv a_1 \ (mod\ n_1) \\ x &\equiv a_2 \ (mod\ n_2) \\ &\vdots \\ x &\equiv a_k \ (mod\ n_k) \end{aligned}$$

The Chinese remainder theorem states that this system of congruences ($x \equiv a_i \ (mod\ n_i)$ for $1 \leq i \leq k$) has a unique solution modulo $N = n_1 \cdot n_2 \ \cdots n_k$ which is given by

$$x = \sum_{i=1}^{k} a_i \, N_i \, y_i \ (mod\ N)$$

where

$$N_i = N/n_i \quad and \quad y_i = {N_i}^{-1} (\ mod\ n_i) \quad for\ 1 \leq i \leq k.$$

2.1.2.1 Division in modular arithmetic

The process of division in modular arithmetic is a bit different from other operations in modular arithmetic. In general, dividing by $a \pmod n$ is allowed when gcd(a, n) = 1,

Theorem 2.16 ([24]) *Let, a, b, c, n are integers. $n \neq 0$, and gcd$(a, n) = 1$. If $ab \equiv ac \ (mod\ n)$, then $b \equiv c \ (mod\ n)$. In other words, if a and n are relatively prime, we can divide both sides of the congruence by a.*

Proof: Given, gcd$(a, n) = 1$. So, there exists integers x, y such that, $ax + ny = 1$.
Multiplying both sides by $b - c$, we get,

$$\begin{aligned} (ax + ny)(b - c) &= b - c \\ \Rightarrow abx - acx + nby - ncy &= b - c \\ \Rightarrow x(ab - ac) + y(b - c)n &= b - c \end{aligned}$$

By assumption, $ab \equiv ac \pmod n$ which means $ab - ac$ is a multiple of n. On the other hand, $y \cdot (b - c) \cdot n$ is a multiple of n. Therefore, certainly $b - c$ is a multiple of n.
Hence, $b \equiv c \pmod n$. □

Example 2.17 $5x + 22 \equiv 2 \pmod{29}$

$$\begin{aligned} & 5x \equiv 2-22 \equiv -20 \pmod{29} \\ \Rightarrow\ & x \equiv -4 \pmod{29} \text{ [Since, } \gcd(5,29)=1\text{, division by 5 is valid]} \\ \Rightarrow\ & x \equiv 25 \pmod{29} \end{aligned}$$

Proposition 2.18 *Let $gcd(a,n)=1$. x and y are two integers such that $ax+ny=1$. Then $ax \equiv 1\ (mod\ n)$. Here, x is the multiplicative inverse of $a\ (mod\ n)$.*

Proof: We have,

$$\begin{aligned} & ax+ny=1 \\ \Rightarrow\ & ax-1=-ny \end{aligned}$$

which represents that $ax-1$ is a multiple of n or n divides $ax-1$. Therefore, $ax \equiv 1 \pmod{n}$. □

Remark 2.19 In some cases, when $\gcd(a,n)=k>1$, it is required to solve the congruence of the form $ax \equiv b \pmod{n}$.

If b can not be divided by $\gcd(a,n)=k$ (i.e. $k \nmid b$), there is no solution. Otherwise, the following method can be applied.

Let $k \mid b$. We consider,$(\frac{a}{k})x \equiv \frac{b}{k} \pmod{\frac{n}{k}}$. Here, $\gcd(\frac{a}{k},\frac{n}{k})=1$ and $\frac{a}{k},\frac{b}{k},\frac{n}{k}$ all are integers.

The above congruence is needed to be solved and a solution named as x_0 will be found. The solutions of the initial congruence $ax \equiv b \pmod{n}$ are x_0, $x_0+\frac{n}{k}$, $x_0+2\cdot\frac{n}{k}$, ..., $x_0+(k-1)\frac{n}{k} \pmod{n}$.

Example 2.20 $16x \equiv 12 \pmod{36}$

Here $\gcd(16,36)=4$ and $4 \mid 12$. If we divide the given congruence by 4, we get, $4x \equiv 3 \pmod{9}$. The multiplicative inverse of $4 \pmod{9}$ is 7.

Hence, $x \equiv 3\cdot 7 \pmod{9} \equiv 21 \equiv 3 \pmod{9}$. So, the solution is $x_0=3$.

Therefore, solutions of the given congruence are $3,\ 3+\frac{36}{4},\ 3+2\cdot\frac{36}{4},\ 3+3\cdot\frac{36}{4}$ or $3 \pmod{36}$, $12 \pmod{36}$, $21 \pmod{36}$, $30 \pmod{36}$.

2.1.3 Euler's Phi Function, Modular Exponentiation

Definition 2.21 (Euler's phi Function) For a positive integer n, the *Euler's phi function* $\phi(n)$ provides the number of positive integers less than n which are relatively prime to n.

$$\phi(n) = \#\{m \in N : 1 \leq m \leq n, gcd(m,n) = 1\}, n \in N.$$

Proposition 2.22 *Let $\phi(p)$ and $\phi(n)$ be Euler's phi function. Then the following statements hold.*

1. *If p is any prime, $\phi(p) = p-1$*
2. *If $n = pq$ where p and q are distinct primes, then, $\phi(n) = (p-1)(q-1)$*
3. *If $n = p^e$, where p is a prime and $e \in N$, $\phi(n) = \phi(p^e) = p^e - p^{e-1}$*

Example 2.23

$$\phi(12) = \phi(3) \cdot \phi(4) = \phi(3) \cdot \phi(2^2) = (3-1) \cdot (2^2 - 2^{2-1}) = 4$$

So there are 4 positive integers less than 12 which are relatively prime to 12 (indeed those numbers are $1, 5, 7, 11$).

Remark 2.24 Applying the Euler's Phi function, it is possible to find out the number of primitive roots of a prime. For any prime p, $\phi(p-1)$ provides the number of primitive roots of p.

Theorem 2.25 (Multiplicativity of Euler's Phi function) *Let $n_1 \ldots n_k \in \mathbb{N}$, gcd$(n_i, n_j) = 1$ (for $i \neq j$). Then, $\phi(n_1) \ldots \phi(n_k) = \phi(n_1 \ldots n_k)$*

Theorem 2.26 (Euler's Theorem) *If gcd $(a,n) = 1$ then, $a^{\phi(n)} \equiv 1 \ (mod\ n)$ where ϕ is the Euler's Phi function.*

Remark 2.27 In the case of n being a prime number, Euler's Theorem is same as Fermat's theorem, since $\phi(n) = n-1$ if n is a prime number.

Theorem 2.28 (Fermat's Little Theorem) *Let p be prime, $a \in \mathbb{Z}$ and gcd$(a,p) = 1$. Then, $a^{p-1} \equiv 1 \ (mod\ p)$.*

Theorem 2.29 (Euler's Phi function for products of two primes) *Let p and q be two distinct primes. Then, $\phi(pq) = (p-1)(q-1)$.*

Theorem 2.30 (Generalised product formula for $\phi(n)$) *For any positive integer n (i.e. $n \in \mathbb{N}$),*

$$\phi(n) = n\prod_{p|n}(1-\frac{1}{p})$$

Proof: Consider the prime factorization of n,

$$n = p_1^{e_1}.....p_k^{e_k}$$

Now,

$$\begin{aligned}\phi(p_1^{e_1}.....p_k^{e_k}) &= \phi(p_1^{e_1}).....\phi(p_k^{e_k})\\ &= (p_1^{e_1}-p_1^{e_1-1}).....(p_k^{e_k}-p_k^{e_k-1})\\ &= p_1^{e_1-1}(p_1-1).....p_k^{e_k-1}(p_k-1)\\ &= p_1^{e_1}(1-\frac{1}{p_1}).....p_k^{e_k}(1-\frac{1}{p_k})\\ &= p_1^{e_1}.....p_k^{e_k}(1-\frac{1}{p_1}).....(1-\frac{1}{p_k})\\ &= n\prod_{i=1}^{k}(1-\frac{1}{p_i})\end{aligned}$$

□

2.1.3.1 Modular Exponentiation

In the field of cryptography, modular exponentiation is very useful and has its own importance. It is performed over a modulus to find the remainder r of the expression $r \equiv a^e \pmod n$. To be noted $0 \leq r < n$. An example with the illustration to find out the remainder using this standard way is given below:

Example 2.31 $7^{280} \pmod{583}$

Now $7^1 \equiv 7 \pmod{583}$

Squaring both sides repeatedly, following expressions are obtained:

$7^2 \equiv 49 \text{ (mod 583)};$ $7^4 \equiv 49^2 \equiv 69 \text{ (mod 583)};$
$7^8 \equiv 69^2 \equiv 97 \text{ (mod 583)};$ $7^{16} \equiv 97^2 \equiv 81 \text{ (mod 583)};$
$7^{32} \equiv 81^2 \equiv 148 \text{ (mod 583)};$ $7^{64} \equiv 148^2 \equiv 333 \text{ (mod 583)};$
$7^{128} \equiv 333^2 \equiv 119 \text{ (mod 583)};$ $7^{256} \equiv 119^2 \equiv 169 \text{ (mod 583)};$

Here $280 = 256 + 16 + 8$. Hence, $7^{280} \equiv 7^{256} \cdot 7^{16} \cdot 7^8 \equiv 169 \cdot 81 \cdot 97 \pmod{583} \equiv 342 \pmod{583}$.

In the example, we didn't have to deal with number greater than 583^2. Generally, in case of computing $a^e \pmod n$, it is not required to deal with numbers greater than n^2. Due to this, modular exponentiation can be performed fast and less memory is required [24].

Remark 2.32

- Euler's theorem: If $a \in \mathbb{Z}_n^*$ then, $a^{\phi(n)} \equiv 1 \pmod n$.
- If $x \equiv y \pmod{\phi(n)}$ where n is the product of any two distinct primes, then, $a^x \equiv a^y \pmod n$ for all integers a; that is when working with modulo such an n, exponents can be reduced modulo $\phi(n)$.
- Let, p be prime. If $x \equiv y \pmod{p-1}$, then, $a^x \equiv a^y \pmod p$ for all integers a. That means, when working on modulo p, where p is a prime, exponents can be reduced modulo $(p-1)$.

2.1.4 Primitive root, Quadratic Residues and Quadratic Nonresidues

Definition 2.33 A *primitive root* modulo p is a number $g \in \{1,\ldots,p-1\}$, whose powers yield all the nonzero congruence classes mod p, where p is a prime i.e.

$$\{g^1, g^2, g^3, \ldots, g^{p-1}\} \pmod p = \{1,2,3,\ldots,p-1\}$$

Remark 2.34 There are $\phi(p-1)$ primitive roots of p and g is a primitive root of p iff g is a generator of the cyclic group $\mathbb{Z}_p^*$.

Example 2.35 The powers of 5 yields all the non zero congruence classes mod 7. So, 5 is a primitive root of 7. However, 4 is not a primitive root of 7.

Definition 2.36 (Quadratic Residues and Quadratic Nonresidues) Let p be an odd prime and $\gcd(a,p) = 1$ i.e, $a \not\equiv 0 \pmod p$. An integer a is called a quadratic residue modulo p if

$$x^2 \equiv a \pmod p.$$

Otherwise, a is called a quadratic nonresidue modulo p. $\mathbb{Z}_p$ is called the field of residues upon division by p. Generally, $\mathbb{Z}_p$ = $\{0,1,\ldots,p-1\}$ where p is being considered as an odd prime.

2.2 Topics on Abstract Algebra

Definition 2.37 A *group* $(G,*)$ is a set of elements with a binary operation * on G which satisfies the following properties:

1. Associativity: The operation of the group is associative that is

$$(x*y)*z = x*(y*z) \text{ for all } x,y,z \in G$$

2. Closure: The group is closed under the operation. If any two elements x and y are in G, then $x*y$ is also in G that is for any $x,y \in G$.

$$(x*y) \in G$$

3. Inverse: There should be an inverse for each element. i.e. $\text{for all } x \in G$, there is an element x^{-1} such that

$$x*x^{-1} = e = x^{-1}*x$$

4. Identity: For any element there exists the identity that is $\text{for all } x \in G$,

$$x*e = x = e*x$$

Definition 2.38 (Commutative Group) A group $(G,*)$ is said to be *commutative (or abelian) group* if it satisfies the following property also:

$$x*y = y*x \text{ for all } x,y \in G,$$

If a group is not commutative, then it is called as non-commutative group or non-abelian group.

Example 2.39 $(\mathbb{Z},+)$ is a group under addition. The set of integers $\mathbb{Z} = \{\ldots,-2,-1,0,1,2,\ldots\}$ with addition operation forms an abelian group, where $e = 0$ is the identity element and the inverse of an element $x \in \mathbb{Z}$ is $-x$.
The set of real numbers without 0 is an abelian group under multiplication where $e = 1$ is the identity element and the inverse of an element x is x^{-1}.

Remark 2.40 A group G is termed as finite if the cardinality of G, i.e. $|G|$ is finite.
$\mathbb{Z}_n$ represents the non negative integers less than n. $\mathbb{Z}_n$ = {0,1,2, ... , n-1}.
$\mathbb{Z}_n^*$, (the multiplicative group of $\mathbb{Z}_n$), represents the set of integers between 1 and n that are relatively prime to n.

$$\mathbb{Z}_n^* = \{a \in Z_n : \gcd(a,n) = 1\}$$

Definition 2.41 The *order of the group* is the cardinality (number of elements) of that group, and denoted as $|G|$.

Definition 2.42 The *order of an element*, $x \in G$ is the smallest positive integer n such that $x^n = e$ where e is the identity element of the group.
If there is no such n, then x is said to have infinite order. It should be noted that all elements of a finite group have finite order.

Definition 2.43 (Cyclic group and generator) A group G is said to be *cyclic* if there exists $g \in G$ such that for each $a \in G$, there is an integer n with $a = g^n$. Such an element g is called a generator.

$$G =< g >= \{g^n : n \in \mathbb{Z}\} \text{ where } g \in G.$$

Let, $g \in \mathbb{Z}_n^*$. If the order of g i.e. ord(g) is $\phi(n)$, then g is termed as generator of $\mathbb{Z}_n^*$. If $\mathbb{Z}_n^*$ consists of a generator, $\mathbb{Z}_n^*$ can be termed as cyclic. A generator is called as primitive element also.

Example 2.44 The group of integers $\mathbb{Z}$ under addition $(+)$ and we claim that the integers can be generated by 1; i.e. $\mathbb{Z}$ = $< 1 >$.

$$< 1 >= \{\ldots, -4, -3, -2, -1, 0, 1, 2, 3, 4, \ldots\}$$

Here, $\mathbb{Z}$ is an infinite cyclic group.

Example 2.45 The group of integers mod n under addition $(+)$ and so, the set of elements is $\{0, 1, 2, 3, 4, \ldots, n-1\}$. The group can be generated by 1. i.e. G = $< 1 >$. Here, G is an finite cyclic group.

The positive and negative multiples of 1 can be observed as below:

$$\cdots -2, -1, 0, 1, 2, \ldots, n-1, n, n+1, n+2, \ldots$$

$$\begin{array}{ll} n \equiv 0 (\text{mod } n); & -1 \equiv n-1 (\text{mod } n); \\ n+1 \equiv 1 (\text{mod } n); & -2 \equiv n-2 (\text{mod } n); \\ n+2 \equiv 2 (\text{mod } n); & -3 \equiv n-3 (\text{mod } n); \\ \ldots\ldots & \ldots\ldots \\ \ldots\ldots & \ldots\ldots \end{array}$$

Integers mod n $(\mathbb{Z}/n\mathbb{Z})$ are finite cyclic groups.

Example 2.46 Let $\mathbb{Z}_{11}^*$ = {1,2,3,4,5,6,7,8,9,10}. Here, $\phi(11) = 10$. So, the order of the group, $|\mathbb{Z}_{11}^*| = 10$. The orders of elements of the group $\mathbb{Z}_{11}^*$ are given below:

$x \in \mathbb{Z}_{11}^*$	1	2	3	4	5	6	7	8	9	10
order of x	1	10	5	5	5	10	10	10	5	2

Table 2.3: Orders of elements of $\mathbb{Z}_{11}^*$

Further, $\phi(11-1) = \phi(10) = \phi(2) \cdot \phi(5) = 4$. 11 has 4 primitive roots (generators) and those are $2, 6, 7$ and 8. Certainly, $\mathbb{Z}_{11}^*$ is a cyclic group.

Remark 2.47 The number of elements in $\mathbb{Z}_n^*$ is said to be the order of $\mathbb{Z}_n^*$ and denoted by $|\mathbb{Z}_n^*|$. Considering the definition of Euler's phi function, we have, $|\mathbb{Z}_n^*| = \phi(n)$.

Remark 2.48 Let $x \in \mathbb{Z}_n^*$, ord$(x) = k$ and $x^l \equiv 1 \pmod{n}$ then $k \mid l$. In particular, $k \mid \phi(n)$.

A few properties of generators are given below [14].

Proposition 2.49 (Properties of generators)

1. *$\mathbb{Z}_n^*$ consists of a generator iff $n = 2, 4, p^e$ or $2p^e$; p is an odd prime and $e \geq 1$. In particular, $\mathbb{Z}_p^*$ has generator if p is prime.*
2. *If g is a generator of $\mathbb{Z}_n^*$, then $\mathbb{Z}_n^* = \{g^i \pmod{n} : 0 \leq i \leq \phi(n) - 1\}$.*
3. *Let g be a generator of $\mathbb{Z}_n^*$. Then $a = g^i \pmod{n}$ is also a generator of $\mathbb{Z}_n^*$ iff gcd $(i, \phi(n)) = 1$. Equivalently, if $\mathbb{Z}_n^*$ is cyclic, then it has $\phi(\phi(n))$ generators.*
4. *g is a generator of $\mathbb{Z}_n^*$ iff $g^{\phi(n)/p} \not\equiv 1 \pmod{n}$ for each prime divisor p of $\phi(n)$.*

Definition 2.50 (Subgroup) Let G be a group. H is said to be a *subgroup* of G, if H is a non-empty subset of G and H is itself a group under the group operation of G.

It is denoted as $H \leq G$ and read as "H is a subgroup of G". It is said to be a *proper subgroup* of G, if H is a subgroup of G and $H \neq G \Rightarrow H < G$.

Every group G has at least two subgroups: G itself, and trivial group {e}.

Example 2.51 Let the multiplicative group $\mathbb{Z}_{11}^* = \{1, 2, ..., 10\}$ of order 10. Here, the group is cyclic and 2 is a generator. The subgroups of $\mathbb{Z}_{11}^*$ and their generators are shown below:

Subgroups	Generators or primitive roots	order
{1}	1	1
{1,10}	10	2
{1,3,4,5,9}	3,4,5,9	5
{1,2,3,...,10}	2,6,7,8	10

Table 2.4: Subgroups of $\mathbb{Z}_{11}^*$

Definition 2.52 A *Ring* is a set R with two operations: $+$ (addition), $\times$ (multiplication) having following properties:

1. For addition, the elements form abelian group i.e. $(R,+)$ is a commutative group.
2. For multiplication, the operation: $\times$ is associative i.e.

$$(x*y)*z = x*(y*z) \ \ \forall\, x,y,z \in \ R$$

Multiplication does not required to be commutative and it is not required for elements to have multiplicative inverse. It is argumentative yet if a ring should contain an identity element 1 for the multiplication operation. For avoiding ambiguity like many others, we also state that if a ring has an element 1, it is termed as ring with identity. Both operations: $+, \times$ are linked by the following distributive property:

$$x \times (y+z) = x \times y + x \times z$$

$$(x+y) \times z = x \times z + y \times z.$$

If a ring is commutative then it is called as commutative ring.

Definition 2.53 An element x of a ring R is called an *invertible element*, if there exist an element $y \in R$ such that $x \times y = 1$.

Definition 2.54 A *field* is a commutative ring with identity in which every non zero element has a multiplicative inverse.

Example 2.55 $\mathbb{Z}/19\mathbb{Z}$ is an example of ring and field also. $\mathbb{Z}/n\mathbb{Z}$ is always a ring but only in case of n being a prime, it will be a field.

Theorem 2.56 (Lagrange theorem [14]) *If G is a finite group and H is a subgroup of G, then $|H|$ divides $|G|$. Hence, if $a \in G$ the order of a divides $|G|$.*

Proposition 2.57 *If $|G| = p$, where p is prime and G is a finite group, then G is cyclic. Moreover, all elements of the group G except the identity are generators of G.*

Proof: Let G be a finite group and $g \in G$. If $|G| = p$ and ord$(g) = k$, then $k \mid p$. Since p is prime, the only possible order of elements of the group G can be 1 and p. On the other hand, only the identity element has order 1. Therefore, all other elements are with order p and termed as generators. □

Lemma 2.58 ([11]) *Assume, the order of an element x of the group G is k i.e, ord(x)=k and $j \mid k$, then ord$(x^j) = \frac{k}{j}$.*

Proof: Consider,

$$(x^j)^{\frac{k}{j}} = x^k = 1.$$

So the order of x^j is at most $\frac{k}{j}$.

Let $i > 0$ such that $(x^j)^i = 1$. Then $x^{ij} = 1$. Further, k is the order of x. Therefore, $ij \geq k$ or $i \geq \frac{k}{j}$. So the order of x^j is exactly $\frac{k}{j}$ i.e, ord$(x^j) = \frac{k}{j}$. □

Proposition 2.59 ([11]) *Let G be a finite group and $x \in G$ and element of order k. Then $x^a = x^b$ iff $a = b$ (mod k)*

Proof: We know that if G is a finite group and $x \in G$ an element of order k, then $x^a = x^{[a \bmod i]}$ for any integer a.
So,

$$x^a = x^{[a \bmod k]} = x^{[b \bmod k]} = x^b.$$

Consider, $a' = a$ mod k and $b' = b$ mod k. Therefore, we can write

$$x^{a'} = x^{b'} \Rightarrow x^{a'} \cdot (x^{b'})^{-1} = 1.$$

If $a' \neq b'$, we consider that $a' > b'$, because a' and b' are smaller than k. Then,

$$x^{a'} \cdot (x^{b'})^{-1} = 1 \Rightarrow x^{a'-b'} = 1$$

which contradicts that k is the order of x. □

Remark 2.60 Assume a finite group G with order i that is $|G| = i$. Then for any element $x \in G$, $x^i = 1$.

Proposition 2.61 ([11]) *Assume i is the order of a finite group G and $g \in G$ with ord$(g) = k$, then $k \mid i$.*

Proof: We know, for a finite group G, if $|G| = i$, then for $g \in G$, $g^i = 1$. As the order of g

is k that is ord$(g) = k$, it can be written that $g^i = g^{[i \bmod k]}$.

If $k \nmid i$, it can be stated that $y = [i \bmod k]$ is a positive integer smaller than k such that $g^y = 1$. But this is not possible because ord$(g) = k$, then it can be concluded that $k \mid i$. □

Example 2.62 Consider a group $\mathbb{Z}_{13}^*$ and the order of the group, $|\mathbb{Z}_{13}^*| = 12$. So, possible orders for the elements are $1, 2, 3, 4, 5, 6$ and 12, because orders of the elements are divisors of order of the group.

3 Discrete Logarithms: Application and Attack

In the field of cryptography, the discrete logarithm has retained a significant role. There is currently no classical efficient method for computing DLP, discrete logarithm problem. Due to intractability of DLP, it has been frequently used to construct different cryptographic systems.

3.1 Discrete Logarithms

Definition 3.1 Let g be a primitive root of p, where p is a prime. For any $a \in \{1,\ldots,p-1\}$, there exists $x \in \{1,\ldots,p-1\}$ such that $g^x = a \pmod p$. Here,

$$g^x = a \pmod p \;\Leftrightarrow\; x = log_g\, a(\bmod p).$$

Here, x is termed as the discrete logarithm of a to base g modulo p. We consider x as unique and the problem of finding x is termed as the discrete logarithm problem.

Example 3.2

$$3^7 \equiv 54 \pmod{79}$$
$$log_3\, 54 \equiv 7 \pmod{79}$$

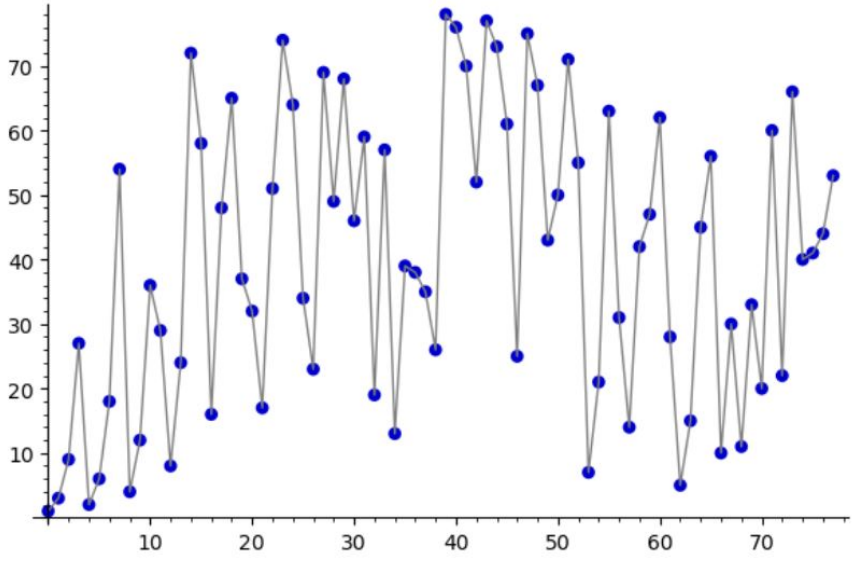

Figure 3.1: Figure for $3^x \pmod{79}$

Here, the figure provides us the values of $3^x \pmod{79}$ for different x that lies between 0 and 77. From the figure, it is clearly understandable how the values fluctuate for different values of x for this discrete logarithm problem.

3.2 Discrete Logarithm Based Cryptography

The area of public-key cryptography is one of the most extensively utilized applications of discrete logarithm problems. Many cryptographic systems, such as ElGamal cryptosystem, various digital signature schemes including ElGamal signature scheme, etc. are based on the hardness of computation of the discrete logarithm problems.

3.2.1 Diffie Hellman Key Exchange

Diffie Hellman key exchange protocol was proposed as the first public key distribution system in 1976 by Whitfield Diffie and Martin E. Hellman in [5] (based on an idea of [15]). The security of the protocol depends on the intractability of DLP. It presented a royal solution of the problem of being agreed on a shared key securely.

Let us consider, Alice and Bob want to share message with each other. In order to keep the message secret, they want to set and exchange a key for the purpose of communication. This can be performed by using Diffie Hellman key exchange method as below:

At the beginning, either Alice or Bob choose a large prime p and a primitive root (or generator) g of prime p. Both agree upon it and make p and g public.

Alice choose a random number, $x \in \{1, \ldots, p-1\}$ and keeps the number secret. She computes $A = g^x \pmod{p}$ and sends A to Bob.

On the other hand, Bob chooses a random number y, where $y \in \{1, \ldots, p-1\}$ and keeps the number secret to himself.

He computes $B = g^y \pmod{p}$ and sends B to Alice. Using their own private key, Alice and Bob can compute the common shared key.

Now, Alice has B and using her secret x, she computes

$$B^x \pmod{p} = (g^y)^x \pmod{p} = g^{xy} \pmod{p}.$$

Further, Bob has A and using his secret y, he computes

$$A^y \pmod{p} = (g^x)^y \pmod{p} = g^{xy} \pmod{p}.$$

So, now Alice and Bob both share $g^{xy} \pmod{p}$.

Now consider, Trudy, an evesdropper, sees A and B and wants to break the key exchange scheme. She can perform it by determining x from $g^x \pmod{p}$ or by determining y from $g^y \pmod{p}$. To do so, Trudy needs to solve the discrete logarithm problem.

It was described that if a primitive root g, prime p and $g^x \pmod{p}$ are given, it is hard to find the value of x. It is easy to break the scheme if the secret key can be found or revealed. To be noted that Diffie Hellman key exchange protocol was proposed to exchange keys securely. Based on the intractability of DLP, there are other methods to send messages securely.

Example 3.3 Let p be a prime and g be a primitive root of p. Alice and Bob agree publicly on $g = 3$, $p = 127$.

Alice chooses a random number $x \in \{1, \ldots, p-1\}$ and keeps the number secret. Let, $x = 45$.

Now, Alice sends $A \equiv 3^{45} \pmod{127} \equiv 95$ to Bob.

Bob chooses a random secret number $y \in \{1, \ldots, p-1\}$. Let, $y = 60$.

Bob sends $B \equiv 3^{60} \pmod{127} \equiv 47$ to Alice.

Now, Alice computes $47^{45} \pmod{127} \equiv (3^{60})^{45} \pmod{127} \equiv 64$.

Bob computes $95^{60} \pmod{127} \equiv (3^{45})^{60} \pmod{127} \equiv 64$

Hence, they both compute the same secret number. Hence, Alice and Bob both share the same key.

3.2.1.1 Man-in-the Middle Attack

While exchanging key, if there is no way for authenticating the attendees, the Diffie -Hellman key exchange might be insecured due to Man-in-Middle Attack.

Let Trudy, an eavesdropper, wants to interrupt in the communication between Alice and Bob and read the messages.

1. At first, Trudy chooses an exponent m.
2. Then she intercepts $A = g^x \pmod{p}$ and $B = g^y \pmod{p}$. She computes and sends $g^m \pmod{p}$ to both of Alice and Bob.

At this point, Alice thinks that she has received it from Bob and Bob thinks that he has received it from Alice.

3. Trudy computes $K_A = (g^x)^m \pmod{p}$ and $K_B = (g^y)^m \pmod{p}$.
4. Alice and Bob do not think that Trudy is in the middle as an eavesdropper. Alice and Bob computes K_A and K_B respectively, where $K_A = g^{mx} \pmod{p}$ and $K_B = g^{my} \pmod{p}$.
5. Trudy intercepts when Alice sends a message, encrypted with K_A to Bob. Then, Trudy deciphers it and can see/read the message. Then, she encrypts the same message or altered message with encryption key K_B and sends to Bob.
6. Bob decrypts the message with K_B. None of Alice and Bob thinks about the communication being insecured and following this way, Trudy can read or access the messages.

However, the Man-in-the Middle Attack can be avoided by authenticating the identities of the participants to each other at the time of forming the key.

Using digital signature is an standard solution since it can assure the participants regarding the origination of the message from the correct person.

3.2.2 ElGamal Cryptosystem

Taher ElGamal proposed ElGamal cryptosystem in [6], based on the difficulty of computing discrete logarithms over finite fields in 1985. It was demonstrated that the way to find the private key x from $g^x \pmod{p}$ to break the cryptosystem is to solve the discrete logarithm problem for which there is no efficient way.

Consider, Alice wants to encrypt a message to send Bob. Bob needs to choose a large prime p and a primitive root g of p. Further Bob chooses a random integer y where $y \in \{1, \ldots, p-1\}$ and keeps the number secret. Then he calculates $B \equiv g^y \pmod{p}$ to declare (p, g, B) as public key. For sending message m, where, $m \in \{1, \ldots, p-1\}$; [m needs to be broken into smaller blocks if it is larger than p].

Alice follows the steps described below:

At first, Alice downloads Bob's public key (p, g, B).

She chooses an integer $x \in \{1, \ldots, p-1\}$ and keeps the number secret.

Then she computes,

$$C_1 \equiv g^x (\text{mod } p) \text{ and } C_2 \equiv B^x \cdot m \pmod{p}$$

Alice sends the cipher-text (C_1, C_2) to Bob.

To recover the message, Bob needs to compute $m \equiv C_1^{-y} \cdot C_2 \pmod{p}$

We can have a look on how the decryption works:

$$C_1^{-y} \cdot C_2 \equiv (g^x)^{-y} \cdot B^x \cdot m \equiv g^{-yx} \cdot (g^y)^x \cdot m \equiv m \pmod{p}$$

Here, the numbers p, g, B are public. If an attacker can determine y, it is also possible to decrypt for the attacker by following the same steps that Bob follows. Hence, $B \equiv g^y \pmod{p}$. The security of y is based on the difficulty of solving DLP.

If Alice wants to send several messages, Alice needs to use different x for each message. It is required because if somehow m_1 is revealed, m_2 can be found easily.

Let us consider, same x has been used for two messages and the cipher text is (C_1, C_2) and (C_1, C_2').

Then,

$$(\frac{C_2}{m_1}) \equiv (B)^x \equiv (\frac{C_2'}{m_2}) \pmod{p}$$

As Trudy already knows about C_1 and C_2, she calculates

$$m_2 \equiv \frac{C_2'}{C_2} \cdot m_1 \pmod{p}.$$

B^x is a random non zero mod p due to x being a random integer. So, $C_2 \equiv B^x \pmod{p}$ is m multiplied by a random integer and if $m \neq 0$ (certainly, m = 0 needs to be avoided), C_2 is random. Hence, C_2 doesn't provide any necessary information to Trudy about m. It is hard to find the integer x from C_1 because to do so, it is required to solve DLP.

3.2.3 ElGamal Signature Scheme

ElGamal's signature scheme was introduced by Taher ElGamal in 1985 in [6]. This digital signature scheme is also based on the intractability of DLP. The idea used in ElGamal signature scheme are the foundations of many popular signature scheme such as Digital Signature Standard (DSS).

Consider, Alice wants to send a message to Bob. She wants to create a digital signature for the sake of the integrity of the message.

At first, Alice chooses a large prime p, a primitive root g of p, where $2 \leq g \leq p-2$. After that she chooses a secret integer x, where $1 \leq x \leq p-2$.

Then she computes $A \equiv g^x \pmod{p}$ and declares (p, g, A) as her public key.

For signing a message $m \in \{1, \ldots, p-1\}$, Alice follows the steps given below:

Alice chooses a random secret integer k such that $1 \leq k \leq p-2$ and $\gcd(k, p-1) = 1$. She computes,

$$t \equiv g^k \pmod{p} \quad \text{and} \quad u \equiv k^{-1}(m - xt) \pmod{p-1}$$

So, the triple (m, t, u) is the signed message and Alice sends it to Bob.

In order to verify the signature, Bob follows:

After receiving (m, t, u), Bob downloads Alice's public key (p, g, A). He checks if $1 \leq t \leq p-1$. If not immediately the signature is rejected by Bob. If the condition is satisfied, he computes,

$$V_1 \equiv A^t \cdot t^u \pmod{p} \quad \text{and} \quad V_2 \equiv g^m \pmod{p}$$

Bob only accepts the signature iff $V_1 \equiv V_2 \pmod{p}$.

Assuming the signature is valid, now, we demonstrate why the verification procedure works. Here, we have,

$$\begin{aligned} u &\equiv k^{-1}(m - xt) \pmod{p-1} \\ k \cdot u &\equiv (m - xt) \pmod{p-1} \\ k \cdot u &\equiv d(p-1) + (m - xt) \pmod{p-1} \end{aligned}$$

Now,

$$\begin{aligned} V_1 &\equiv A^t \cdot t^u \pmod{p} \\ &\equiv (g^x)^t \cdot (g^k)^u \pmod{p} \\ &\equiv g^{xt+ku} \pmod{p} \end{aligned}$$

Since, a congruence mod $p-1$ in the exponent yields an overall congruence mod p,

$$\begin{aligned} V_1 &\equiv g^{xt+d(p-1)+(m-xt)} \pmod{p} \\ &\equiv g^{m+d(p-1)} \pmod{p} \\ &\equiv g^m \cdot (g^{(p-1)})^d \pmod{p} \\ &\equiv g^m \cdot 1 \pmod{p} \text{ [Applying Fermat's little theorem]} \\ &\equiv V_2 \pmod{p} \end{aligned}$$

It is must to keep x secret because if somehow the attacker can reveal x, he or she can perform the signing procedure and forge the signature of Alice on any document.

Even if Trudy finds another message m, it is not possible for her to find out the appropriate u, due to lack of x. She can have a try to choose or find such an u for which the verification equation is satisfied which means it is required to choose such an u that satisfies:

$$\begin{aligned} A^t \cdot t^u &\equiv g^m \pmod{p} \\ t^u &\equiv A^{-t} \cdot g^m \pmod{p} \end{aligned}$$

However, to perform so, it is needed to solve discrete logarithm problem.

Hence, ElGamal signature scheme's security is based on the computation of solving discrete log mod p.

To be noted that Different random session key k needs to be chosen for sending message every time. Let same k has been used by Alice for two different messages m and m'. In that case Trudy can realize the fact of using same k since there is no change of value of t. Then,

$$\begin{aligned} u_1 &\equiv k^{-1}(m - xt) \pmod{p-1} \\ u_2 &\equiv k^{-1}(m' - xt) \pmod{p-1} \end{aligned}$$

Hence, Trudy computes,

$$\begin{aligned} u_1 \cdot k - m \equiv -x \equiv u_2 \cdot k - m' \pmod{p-1} \\ (u_1 - u_2) \cdot k \equiv m - m' \pmod{p-1} \end{aligned}$$

Assume $\gcd(u_1 - u_2, p-1) = c$.

So there are c solutions to the congruences.

Generally, c is a small number and therefore there might not be a lot of possible values to k. Until Trudy finds the value of t, she computes g^k for each possible k. Finally she reveals k.

Now, in order to obtain x, she computes,

$$xt \equiv m - k \cdot u_1 \pmod{p-1}$$

Assume, $\gcd(t, p-1) = d$.

So, there are d solutions to the congruences.

Following the above strategy, she computes g^x for each possible x until the value of A is found. At one stage she reveals x and become able to break the system or method completely and can forge the signature of Alice.

The security or safety of the system is dependent on keeping x secret and it is hard to find x from (p,g,A) since to do so, it is needed to solve DLP.

Example 3.4 Let Alice chooses a prime, p = 10103, a primitive root, $g = 5$.

She chooses a secret integer x = 234. Now, Alice computes,

$$A \equiv g^x \pmod{p} \equiv 5^{234} \equiv 283 \pmod{10103}.$$

Alice declares $(10103, 5, 283)$ as her public key. Alice wants to send message $m = 1213$, where it is a hashed value of the message. She selects a secret integer $k = 345$ whereas gcd$(345, 10102) = 1$.

Now, Alice calculates,

$$\begin{aligned} t &\equiv g^k \pmod{p} \\ &\equiv 5^{345} \pmod{10103} \\ &\equiv 2996 \pmod{10103} \end{aligned}$$

She also calculates,

$$\begin{aligned} u &\equiv k^{-1}(m - xt) \pmod{p-1} \\ &\equiv 345^{-1}(1213 - 234 \cdot 2996) \pmod{10102} \\ &\equiv 937 \cdot (-699851) \pmod{10102} \\ &\equiv 841 \pmod{10102} \end{aligned}$$

Alice sends the triple $(1213, 2996, 841)$ to Bob.

After that, Bob computes,

$$\begin{aligned} V_1 &\equiv A^t \cdot t^u \pmod{p} \\ &\equiv 283^{2996} \cdot 2996^{841} \pmod{10103} \\ &\equiv 9795 \pmod{10103} \end{aligned}$$

Further, he calculates,

$$\begin{aligned} V_2 &\equiv g^m \pmod{p} \\ &\equiv 5^{1213} \pmod{10103} \\ &\equiv 9795 \pmod{10103} \end{aligned}$$

Since V_1 = V_2, Bob accepts the signature.

Example 3.5 Let, Alice chooses a prime, p = 12113, a primitive root, g=5,
She selects a secret integer x = 456.

She computes, $A \equiv g^x \pmod{p} \equiv 5^{456} \pmod{12113} \equiv 7442 \pmod{12113}$
Alice declares $(12113, 5, 7442)$ as her public key.

Let $m = 2324$
Alice selects a random key k to sign the message and keeps the number secret.

Then she calculates,

$$\begin{aligned} t &\equiv g^k \pmod{p} \\ &\equiv 2147 \pmod{12113} \end{aligned}$$

She also calculates,

$$\begin{aligned} u_1 &\equiv k^{-1}(m - xt) \pmod{p-1} \\ &\equiv 3596 \pmod{12112} \end{aligned}$$

So, in the first scenario, the generated signed message is $(2324, 2147, 3596)$ which was sent to Bob by Alice.

Now, assume Alice wants to send second message, $m' = 1819$ using same k.

Following the same way, value of t will be same i.e. 2147 and she computes,

$$\begin{aligned} u_2 &\equiv k^{-1}(m' - xt) \pmod{p-1} \\ &\equiv 2239 \pmod{12112} \end{aligned}$$

So, in the second phase, Alice sends the triple $(1819, 2147, 2239)$ to Bob.
Since the value of t is unchanged, Trudy realize that same k for two different messages has been used.

Then, Trudy computes,

$$\begin{aligned} k \cdot (u_1 - u_2) &\equiv (m - m') \pmod{p-1} \\ k \cdot (3596 - 2239) &\equiv (2324 - 1819) \pmod{12112} \\ 1357 \cdot k &\equiv 505 \pmod{12112} \end{aligned}$$

Here, $\gcd(1357, 12112) = 1$.

So, there is only one solution.

$$\begin{aligned} 1357 \cdot k &\equiv 505 \pmod{12112} \\ k &\equiv 4677 \cdot 505 \pmod{12112} \\ k &\equiv 45 \pmod{12112} \end{aligned}$$

Now,

$$\begin{aligned} t &\equiv g^k \pmod{p} \\ &\equiv 5^{45} \pmod{12113} \\ &\equiv 2147 \pmod{12113} \end{aligned}$$

So, Trudy finds the value of t as a correct one and concludes $k = 45$.

Now,

$$(u_1 \cdot k) \equiv (m - xt) \pmod{p-1}$$

Trudy calculates,

$$\begin{aligned} x \cdot 2147 \equiv x \cdot t \equiv m - u_1 \cdot k \equiv (2324 - 3596 \cdot 45) \pmod{12112} \\ \Rightarrow x \cdot 2147 \equiv 10072 \pmod{12112} \end{aligned}$$

$\gcd(2147, 12112) = 1$; So, there is only one solution.

$$\begin{aligned} x \cdot 2147 &\equiv 10072 \pmod{12112} \\ \Rightarrow x &\equiv 2155 \cdot 10072 \pmod{12112} \\ \Rightarrow x &\equiv 456 \pmod{12112} \end{aligned}$$

Trudy checks $5^{456} \pmod{12113} \equiv 7442$.

Since, for $x = 456$, Trudy obtains the value of A. She confirms herself that $x = 456$.

Since, Trudy knows the private key of Alice, she can forge Alice's signature.

3.3 Algorithms For Discrete Logarithm Problem

Algorithms for solving the discrete logarithm problem can be divided into two types: those that suit for any group (often referred to as generic algorithms) and those that work for certain groups. Keeping the cryptographic significance in mind, we start with the computation of discrete logarithm problems in the cyclic group $\mathbb{Z}_p^*$. There are several methods to solve discrete logarithms problems and some of those will be demonstrated here.

3.3.1 Baby-Step Giant-Step Algorithm

In 1971, Shank's idea to calculate the order of an element of a group which is used to compute discrete logarithms, was published in [21].

Now, this powerful method of computing discrete logarithm will be described here. Suppose, a prime p, a primitive root g and $a = g^x \pmod p$ are given. We would like to find out the value of x. The following steps are considered to determine the value of x:

Step 1: Let $m = \lceil \sqrt{p} \rceil$

Using Euclid theorem, we write, $x = im + j$ for $i \in \{0, 1, \ldots, m-1\}$ and $j \in \{0, 1, \ldots, m-1\}$. Hence,

$$a = g^x (\text{mod } p) = g^{im+j} (\text{mod } p)$$

Therefore,

$$g^j = ag^{-im} \pmod p \tag{3.1}$$

If we can find out the value of i and j, which satisfies (3.1) then the value of x can be found easily since $x = im + j$.

Step 2: We compute $g^j \pmod p$ for each $j = 0, 1, \ldots, m-1$ and store the results in a table.

Step 3: We start computing, $a \cdot g^{-im} \pmod p$ for all $i = 0, 1, \ldots, m-1$ and compare the obtained results with all the computed values of $g^j \pmod p$ stored in the table. We stop if a match is found.

Step 4: If a match is found in step 3, then the values of i and j are also found which satisfies (3.1) as well. We compute, $x = im + j$. Hence, the value of x will be determined.

Remark 3.6 In the above method, $a \cdot g^{-im} \pmod p$ represents the Giant steps and $g^j \pmod p$ represents the baby steps.

3.3.1.1 Time and space complexities of Baby-step Giant-step's algorithm

It is required to compute about $\sqrt{p}$ values which are stored in a table. Therefore, for the baby steps it is needed to compute $\sqrt{p}$ group operations. Further, it requires one group operation for computing each giant step.

The time complexity of the algorithm is $\mathcal{O}(\sqrt{p}\ log\ p)$ and needs space of $\mathcal{O}(\sqrt{p})$.

Assuming that group multiplication takes longer than log p comparisons, it can be expressed that the running time of the baby-step giant-step algorithm is $\mathcal{O}(\sqrt{p}$ group multiplications. In general, the method may find a matching after half of the giant steps are computed on average. However, in the worst case it is needed to compute m which is around $\sqrt{p}$ giant steps. Therefore, in the worst case, the time complexity will be $\mathcal{O}(m^2) = \mathcal{O}(p)$ [22] [14].

Example 3.7 Let $g = 7,\ p = 151$. We want to solve the discrete log of $a = g^x$ (mod $p) = 73$.

At first, we select m = $\lceil\sqrt{151}\ \rceil$ = 13 and note that $g^{-m} = 7^{-13} = 35$ (mod p).

Then, we compute g^j (mod p) for each $j \in \{0, 1, \ldots, m-1\}$ and store the output in table-1.

Table-1: Baby Step Computation

Baby step j	0	1	2	3	4	5	6	7	8	9	10	11	12
$7^j \pmod{151}$	1	7	49	41	136	46	20	140	74	65	2	14	98

Now, for $i = 0, 1, \ldots m-1$,

$$a \cdot g^{-im} \equiv 73 \cdot 7^{-13i} \equiv 73 \cdot 35^i \ (\text{mod } 151).$$

We need to compute the above expression until we obtain a value which also exists in the second row the of previously computed table-1.

$$73 \cdot 35^0 \equiv 73 \text{ (mod } 151)$$
$$73 \cdot 35^1 \equiv 139 \text{ (mod } 151)$$
$$73 \cdot 35^2 \equiv 33 \text{ (mod } 151)$$
$$73 \cdot 35^3 \equiv 98 \text{ (mod } 151)$$

Such a match has been found for $i = 3$. For $i = 3$, we get $73 \cdot 7^{(-13)\cdot 3} \equiv 73 \cdot 35^3 \equiv 98 \pmod{151}$ which is already in the thirteenth column(for $i = 12$) and the second row of table-1.

So, $g^j = ag^{-im} \pmod{p}$ holds for $i = 3$, $j = 12$ and $m = 13$ and $7^{12} = 73 \cdot 7^{-3\cdot 13} \pmod{151}$.

Therefore, 7^{51} = 73 (mod 151) and $x = 51$ is the solution of the discrete logarithm problem.

3.3.2 Index Calculus Method

Index Calculus is known as a powerful method for computing discrete logarithm. A subexponential algorithm for DLP was introduced by Adleman in 1979 in [1]. There are several variants of Adleman's index calculus for DLP in $\mathbb{Z}_p^*$. However, the basic ideas of this method was due to Kraitchik (1922).

Let, a prime p, a primitive root g and a = g^x (mod p) are given. We are required to find out the value of x and to do so the following steps are considered:

Step 1: At first, choose a bound B and let S = $\{p_0, p_1, p_2, \ldots, p_{n-1}\}$ be the set of primes in the factor base where the elements of S are less than B.

Step 2: Choose a random integer k $(0 \leq k \leq p-1\ )$ and compute g^k.

Step 3: Try to find such k where g^k can be represented as a product of elements in the factor base S:

$$g^k = \prod_{i=0}^{n-1} p_i^{c_i},\ c_i \geq 0 \tag{3.2}$$

If successful, take logarithms with base g in both sides of equation (3.2) to get a linear relation

$$k = \sum_{i=0}^{n-1} c_i\ log_g\ p_i \pmod{p-1} \tag{3.3}$$

Step 4: Repeat above step 2 and step 3 until we obtain $n + c$ relations (n is the number of unknown, c is a small positive integer) of the form (3.3).

Step 5: Solve the linear system of n+c equations of the form (3.3).
Then, the values of $log_g\ p_i\ (0 \leq i \leq n-1)$ are found.

Step 6: Choose a random integer k, $(0 \leq k \leq p-2)$ and find the values of $a \cdot g^k$.

Step 7: Try to find such $a \cdot g^k$ which can be written as a product of elements in S.

$$a \cdot g^k = \prod_{i=0}^{n-1} p_i^{d_i}, \ d_i \geq 0 \tag{3.4}$$

If the attempt is not successful, step 6 needs to be repeated.

In case of successful attempt, we take logarithm with base g on both sides of equation (3.4) and obtain

$$log_g\ a = (\sum_{i=0}^{n-1}\ d_i\ log_g\ p_i - k)\ (\text{mod}\ p-1)$$

Thus, we compute,

$$x = (\sum_{i=0}^{n-1}\ d_i\ log_g\ p_i - k)\ (\text{mod}\ p-1)$$

Hence, the desired value of a for $a = g^x$ (mod p) has been found.

3.3.2.1 Time complexity for Index Calculus algorithm

The size n of the factor base should be carefully chosen to optimize the index-calculus algorithm's execution time. The heaviest part of the Index Calculus method is the computation of logarithms of the elements in the factor base. It is a subexponential time algorithm and the expected running time is $\mathcal{O}(exp(c\sqrt{log\ (p)\ (log\ log\ (p))}))$.

Example 3.8 Let $g = 3$, $p = 33331$, and a = 3^x = 15288 (mod 33331) are given. We want to find out the value of x.

As our factor base, we choose $\{2,3,5,7,11,13\}$ as the set of primes. Now, we will try to find such k where g^k (mod p) can be represented as the product of primes that are in the factor base. Out of several attempts, only successful attempts are given below:

$$\begin{aligned}
3^{1484} &\equiv 180 \equiv 2^2 \cdot 3^2 \cdot 5\ (\text{mod}\ 33331);\\
3^{3082} &\equiv 22 \equiv 2 \cdot 11\ (\text{mod}\ 33331);\\
3^{3083} &\equiv 66 \equiv 2 \cdot 3 \cdot 11\ (\text{mod}\ 33331);\\
3^{5506} &\equiv 105 \equiv 3 \cdot 5 \cdot 7\ (\text{mod}\ 33331);\\
3^{17627} &\equiv 1155 \equiv 3 \cdot 5 \cdot 7 \cdot 11\ (\text{mod}\ 33331);\\
3^{8588} &\equiv 2310 \equiv 2 \cdot 3 \cdot 5 \cdot 7 \cdot 11\ (\text{mod}\ 33331);\\
3^{5411} &\equiv 2535 \equiv 3 \cdot 5 \cdot 13^2\ (\text{mod}\ 33331);
\end{aligned}$$

$$3^{11010} \equiv 1225 \equiv 5^2 \cdot 7^2 \pmod{33331};$$

Now, we obtain the following linear relation by taking logarithms with base $g = 3$, in both sides of the above relations,

$$\begin{aligned}
1484 &\equiv 2\ log_3\ 2\ +\ 2\ log_3\ 3\ +\ log_3\ 5 \pmod{33330} \\
3082 &\equiv log_3\ 2\ +\ 2\ log_3\ 11 \pmod{33330} \\
3083 &\equiv\ log_3\ 2\ +\ log_3\ 3\ +\ log_3\ 11 \pmod{33330} \\
5506 &\equiv\ log_3\ 3\ +\ log_3\ 5\ +\ log_3\ 7 \pmod{33330} \\
17627 &\equiv\ log_3\ 3\ +\ log_3\ 5\ +\ log_3\ 7\ +\ log_3\ 11\ \pmod{33330} \\
8588 &\equiv\ log_3\ 2\ +\ log_3\ 3\ +\ log_3\ 5\ +\ log_3\ 7\ +\ log_3\ 11\ \pmod{33330} \\
5411 &\equiv\ log_3\ 3\ +\ log_3\ 5\ +\ 2\ log_3\ 13 \pmod{33330} \\
11010 &\equiv\ 2\ log_3\ 5\ +\ 2\ log_3\ 7 \pmod{33330}
\end{aligned}$$

After solving above system of linear equations, we obtain $log_3\ 2 = 24291;\ log_3\ 3 = 1;$ $log_3\ 5 = 19560;\ log_3\ 7 = 19275;\ log_3\ 11 = 12121;\ log_3\ 13 = 26255.$

Then, we continue to choose k arbitrarily until we get a value of $15288 \cdot 3^k \pmod{33331}$ that factors completely over the factor base.

When we consider $k = 18$, we obtain,

$$15288 \cdot 3^{18} = 100 \pmod{33331} \tag{3.5}$$

where $100 = 2^2 \cdot 5^2$ factors completely over the factor base.

Now, we take log with base 3 on both sides of (3.5) and obtain,

$$log_3\ 15288 + 18\ log_3\ 3 \equiv\ 2\ log_3\ 2 + 2\ log_3\ 5 \pmod{33330}$$

Now, we replace the solved values for the logarithms of the primes in the factor base and obtain,

$$log_3\ 15288 = 2 \cdot 24291 + 2 \cdot 19560 - 18 \pmod{33330} = 21024 \pmod{33330}$$

which implies that

$$3^{21024} = 15288 \pmod{33331}$$

Therefore, $x = 21024$. Hence, the DLP is solved using Index Calculus method.

3.3.3 Pollard Kangaroo Method

Pollard's Kangaroo method to compute discrete logarithms in arbitrary cyclic group was published in 2000 in [19] originated from [20]. To implement the algorithm it is necessary to know the interval where solution lies.

Let $\mathbb{Z}_p^*$ be a finite cyclic group, g is a primitive root of prime p. The goal is to solve $g^x = a \bmod p$ where $x \in \mathbb{Z}^+$ and $x < p-1$. The Kangaroo method was structured in such a way where the solution lies in interval DLP (: given an element $a \in G$, an integer x is needed to be determined such that $l \leq x < u$ and $a = g^x$). It is considered that the values of l and u are known and $w = u - l$.

We choose a small set of positive integers $S = (s_1 \dots s_k)$. Indeed, elements of this set represents the jumps of kangaroos. Powers of two are good choices though it was not ensured that those are the best choice. Therefore, other integers can be picked up as well. Here the repetition is allowed and elements are placed in non decreasing order. To be noted that the mean of the set, is comparable to $\sqrt{w}$. The interval, we considered earlier, can be increased but then more time will be required for solving the problem. On the other side,if the interval is decreased, the method might experience more failures.Then, it is required to consider a pseudo-random walk. We are required to determine the value of x and to do so the following steps are considered.

Step 1: Let $w = u - l$. We choose a small set of positive integers $S = (s_1 \dots s_k)$. For each element $k \in \mathbb{Z}_p^*$, we would like to have a pair of random integer. We consider the mapping $f : \mathbb{Z}_p^* \to S$ where S is the set of random integers.

Step 2: Let $m = \lfloor \sqrt{p-1} \rfloor + 1$. Kangaroo can be considered as a sequence of element of G of the form:

$$x_{i+1} \equiv x_i \cdot g^{f(x_i)}, \text{ for } i \geq 0,\ i \in \{0, 1, \dots, m-1\}$$

However, the starting point is considered at x_0. The first value of the first list, the value of x_0 is considered as the primitive root raised to the upper bound of the interval DLP. So $x_0 \equiv g^u$ and for other cases $x_{i+1} \equiv x_i \cdot g^{f(x_i)},\ i \geq 0$.

Step 3: We compute

$$d_M = \sum_{i=0}^{m-1} f(x_i)$$

which is the summation of random positive integers that were chosen and mapped into through pseudorandom mapping. Thus we make the first list. Now, a second list needs to be prepared.

Step 4: Let the first step, $y_0 = a$ and for all others,

$$y_{i+1} = y_i \cdot g^{f(y_i)}$$

We calculate

$$d = \sum_{i=0}^{m-1} f(y_i)$$

We continue the computation of y_i until the following one of the conditions of the following step is reached.

Step 5: We stop computing y_{i+1} if the algorithm can determine $y_j = x_m$ for some j and in this case, the solution of DLP can be found by calculating

$$x = u + d_M - d$$

In the second case, we need to check if $d > u - l + d_M$. If so, it is considered that the algorithm fails to find x and it is needed to start from the beginning again with a different pseudorandom walk.

3.3.3.1 Time complexity of Pollard Kangaroo method

In the Kangaroo method, the interval where the solution lies should be known. Therefore, the running time is $\mathcal{O}(\sqrt{w})$ where w is the width of the interval. Van Oorschat and Weiener represented the efficient parallel version of Kangaroo method obtaining linear speed up. Let, q is the largest prime factor of order of the group. Then, the running time is $\mathcal{O}(\frac{\sqrt{q}}{P})$ and $\mathcal{O}(\frac{\sqrt{w}}{P})$ with P processor [19].

Example 3.9 Let us consider a small example. Let $\mathbb{Z}_{17}^*$ be a finite cyclic group, with a primitive root $g = 3$ of prime p. We are required to determine x for $3^x = 12 \pmod{17}$. So $g = 3, a = 12$. Assume x lies in $g \leq x < 16$, $u = 16, l = 8$. We consider the mapping as follows:

$f(1) = 1;$	$f(2) = 2;$	$f(3) = 4;$	$f(4) = 8;$
$f(5) = 1;$	$f(6) = 8;$	$f(7) = 4;$	$f(8) = 8;$
$f(9) = 2;$	$f(10) = 4;$	$f(11) = 1;$	$f(12) = 8;$
$f(13) = 4;$	$f(14) = 2;$	$f(15) = 1;$	$f(16) = 1;$

Now, $m = \lfloor \sqrt{17-1} \rfloor + 1 = 5$. We have, $x_0 \equiv g^u \equiv 3^{16} \equiv 1 \pmod{17}$.

We need to compute

$$x_{i+1} = x_i \cdot g^{f(x_i)} \text{ for } i \in \{0,1,2,3,4\}.$$

So,

$$x_1 \equiv x_0 \cdot g^{f(x_0)} \equiv 1 \cdot 3^{f(1)} \equiv 1.3^1 \equiv 3 \pmod{17};$$
$$x_2 \equiv x_1 \cdot g^{f(x_1)} \equiv 3 \cdot 3^{f(3)} \equiv 3.3^4 \equiv 5 \pmod{17};$$
$$x_3 \equiv x_2 \cdot g^{f(x_2)} \equiv 5 \cdot 3^{f(5)} \equiv 5.3^1 \equiv 15 \pmod{17};$$
$$x_4 \equiv x_3 \cdot g^{f(x_3)} \equiv 15 \cdot 3^{f(15)} \equiv 15 \cdot 3^1 \equiv 11 \pmod{17};$$
$$x_5 \equiv x_4 \cdot g^{f(x_4)} \equiv 11 \cdot 3^{f(11)} \equiv 11.3^1 \equiv 16 \pmod{17};$$

Here,

$$d = f(x_0) + f(x_1) + f(x_2) + f(x_3) + f(x_4) = 1 + 4 + 1 + 1 + 1 = 8.$$

Now, we start computing

$$y_{i+1} = y_i \cdot g^{f(y_i)} \text{ for } i \in \{0,1,2,3,4\}$$

We consider that $y_0 = 12 = a$. Now, $y_0 = 12$ and

$$y_1 \equiv 12 \cdot 3^{f(12)} \equiv 12 \cdot 3^8 \equiv 5 \pmod{17}$$

We can stop here since,

$$x_2 \equiv 5 \equiv y_1 \pmod{17}$$

So, we have found the matching. Now, $d = f(y_0) = 8$. So,

$$x = u + d_M - 8 = 16 + 5 - 8 = 13$$

Since, $3^{13} \equiv 12 \pmod{17}$, the solution $x = 13$ is correct.

3.3.4 Silver Pohlig Hellman Algorithm

Silver Pohlig Hellman Algorithm was introduced by Pohlig and Hellman in 1978 in [18] to compute discrete logarithm over the finite field $G(\mathbb{F}_p)$.

The algorithm to compute discrete logarithms mod p would be more comprehensible if an special case $p = 2^n + 1$ is considered at first.

3.3.4.1 Silver Pohlig Hellman algorithm for special case: $p = 2^n + 1$

Let g be a primitive element of a finite field $G(\mathbb{F}_p)$. It is required to find the solution (x) for

$$x = \log_g a \pmod{p} \Rightarrow a \equiv g^x \pmod{p}$$

where a, g, p are given. We assume that $0 \leq x \leq p - 2$.

Let $x_0, x_1, \ldots, x_{n-1}$ be the binary expansion of x. When $p = 2^n + 1$, the unknown x can

be found by binary expansion,

$$\begin{aligned} x &= x_0 + x_1 \cdot 2 + \cdots + x_{n-1} \cdot 2^{n-1} \\ &= \sum_{i=0}^{n-1} x_i \cdot 2^i;\ x_i \in \{0,1\},\ 0 \le i \le n-1. \end{aligned}$$

Therefore, if x_i,s can be found, the value of x can be determined simply. Since g is a primitive element of $G(\mathbb{F}_p)$, $g^{p-1} \equiv 1 \pmod{p}$ and $g^i \neq 1 \pmod{p}$ for $0 < i < p-1$.

The least significant bit x_0 of x can be found by raising a to the power of $\frac{(p-1)}{2}$ and apply the rule

$$a^{\frac{(p-1)}{2}} \pmod{p} = \begin{cases} +1 & \text{if } x_0 = 0 \\ -1 & \text{if } x_0 = 1 \end{cases}$$

We also note that $g^{\frac{(p-1)}{2}} \equiv -1 \pmod{p}$ since the square of $g^{\frac{(p-1)}{2}}$ is 1 but $g^{\frac{(p-1)}{2}} \neq 1$.
We consider,

$$a^{\frac{(p-1)}{2}} \equiv (g^x)^{\frac{(p-1)}{2}} \equiv g^{(x_0 + x_1 \cdot 2 + \ldots\ldots + x_{n-1} \cdot 2^{n-1}) \cdot \frac{(p-1)}{2}}$$

Therefore evaluation of the following congruence will yield x_0.

$$a^{\frac{(p-1)}{2}} \equiv (-1)^x \pmod{p}$$

Then we compute

$$a_1 \equiv a \cdot g^{-x_0} \equiv g^{s_1} \pmod{p}$$

where

$$s_1 = \sum_{i=1}^{n-1} x_i \cdot 2^i$$

Here s_1 is a multiple of 4 if $x_1 = 0$ and s_1 is a multiple of 2 but not 4 if $x_1 = 1$.
Raising a_1 to the power of $\frac{(p-1)}{4}$ and applying the following rule, bit x_1 of x can be determined.

$$a^{\frac{(p-1)}{4}} \equiv g^{\frac{x(p-1)}{2}} \equiv \begin{cases} +1 & \text{if } x_1 = 0 \\ -1 & \text{if } x_1 = 1 \end{cases}$$

Then, we compute $a_2 = a_1 \cdot g^{-2x_1}$ and sequentially we obtain x_2 from $a_2^{\frac{(p-1)}{8}}$. We continue the same process until x_{n-1} has been found. Finally using $x_0, x_1, \ldots, x_{n-1}$ it can be determined the value of x.

Remark 3.10 In the next section, the above algorithms for $p = 2^n + 1$ will be generalized to arbitrary primes since $2^{16} + 1$ is the largest known prime of the form $2^n + 1$.

Example 3.11 Let us solve a small problem $x = log_g\ 13 \pmod{17}$ where the primitive

element, $g = 5$.

Let $a = 13$, $p = 17 = 2^4 + 1$. We consider the binary expansion

$$x = x_0 + 2 \cdot x_1 + 4 \cdot x_2 + 8 \cdot x_3$$

Here,

$$a^{\frac{(p-1)}{2}} \equiv (13)^8 \equiv 1 \pmod{17}$$

Using the formula

$$a^{\frac{(p-1)}{2}} \equiv \begin{cases} +1 & \text{if } x_0 = 0 \\ -1 & \text{if } x_0 = 1 \end{cases}$$

We get $x_0 = 0$. Then

$$a_1 \equiv a \cdot g^{-x_0} \equiv 13 \cdot (5)^0 \equiv 13 \pmod{17}$$

Accordingly we compute

$$a_1^{\frac{(p-1)}{4}} \equiv (13)^4 \equiv 1 \pmod{17}$$

Hence, $x_1 = 0$. Further

$$a_2 \equiv a_1 \cdot g^{-2x_1} \equiv 13 \pmod{17}$$

So,

$$a_2^{\frac{(p-1)}{8}} \equiv (13)^2 \equiv -1 \pmod{17}$$

Therefore, $x_2 = 1$. Finally

$$a_3 \equiv a_2 \cdot g^{-4x_2} \equiv 13 \cdot (5)^{-4} \pmod{17} \equiv 1 \pmod{17}$$

$$a_3^{\frac{(p-1)}{16}} \equiv 1 \pmod{17}$$

So $x_3 = 0$. Hence $x = 4$ which satisfies

$$4 \equiv x \equiv \log_5 13 \pmod{17}.$$

3.3.4.2 Silver Pohlig Hellman algorithm for arbitrary primes

Now the above algorithms for $p = 2^n + 1$ will be generalized to arbitrary primes. Let p be a prime and g be a primitive root of p. It is required to find the solution (x) for

$$x = \log_g a \pmod{p}$$
$$\Rightarrow g^x = a \pmod{p}$$

Initially, we consider the prime factorization of the order of the group.
Since p is prime,

$$\phi(p) = p-1 = \prod_{i=1}^{k} p_i^{n_i} = p_1^{n_1} \cdot p_2^{n_2} \cdot \cdots \cdot p_k^{n_k}$$

where $p_i < p_{i+1}$, p_i are distinct primes and n_i are positive integers.

The value of x for $x \pmod{{p_i}^{n_i}}$ for $i = 1,2,\ldots,k$; needs to be computed for each ${p_i}^{n_i}$ and then with the help of Chinese Remainder Theorem, the congruences are combined and the solution is obtained.

$$x \pmod{\prod_{i=1}^{k} {p_i}^{n_i}} \equiv x \pmod{p-1} \equiv x, \text{ Since } 0 \leq x \leq p-2.$$

The expansion of $x \pmod{{p_i}^{n_i}}$ is considered as follows:

$$x \pmod{{p_i}^{n_i}} = \sum_{j=0}^{n_i-1} x_j {p_i}^j$$

Therefore,

$$x = x_0 + x_1 p_i + x_2 {p_i}^2 + \cdots + x_{n_i-1}\ {p_i}^{n_i-1} \text{ where } 0 \leq x_n \leq p_i - 1.$$

Now, we will determine the values of $x_0, x_1, \ldots, x_{n_i-1}$ sequentially.
It should be noted that

$$x(\frac{p-1}{p_i}) \equiv x_0(\frac{p-1}{p_i}) + (p-1)(x_1 + x_2 p_i + x_3 {p_i}^2 + \ldots) \pmod{p}$$
$$\Rightarrow x(\frac{p-1}{p_i}) \equiv x_0(\frac{p-1}{p_i}) + (p-1)\cdot\omega \pmod{p} \text{ ; where } \omega \text{ is an integer.}$$

We have, $a \equiv g^x$. Raising both sides to the power of $(\frac{p-1}{p_i})$, we get,

$$\begin{aligned} a^{\frac{(p-1)}{p_i}} &\equiv g^{\frac{x(p-1)}{p_i}} \\ &\equiv g^{\frac{x_0(p-1)}{p_i} + (p-1)\cdot\omega} \\ &\equiv g^{\frac{x_0(p-1)}{p_i}} \cdot (g^{p-1})^{\omega} \\ &\equiv g^{\frac{x_0(p-1)}{p_i}} \quad \text{[Applying Fermat's Little theorem]} \end{aligned}$$

Hence, $a^{\frac{(p-1)}{p_i}} \equiv g^{\frac{x_0(p-1)}{p_i}} \pmod{p}$.

The values of a and g are known. So, x_0 can be found by looking at the powers or by applying trial and error method.

We need to find a such k where $0 \leq k \leq p_i - 1$ for which

$$a^{\frac{(p-1)}{p_i}} \equiv g^{\frac{k(p-1)}{p_i}} \bmod p$$

is satisfied and then $x_0 = k$. We know,

$$g^{s_1} \equiv g^{s_2} (\text{mod } p) \Leftrightarrow s_1 \equiv s_2 (\text{mod } p-1).$$

As the exponents $\frac{k(p-1)}{p_i}$ are distinct modulo $p-1$, there is a unique k that satisfies the condition.

Extending the above approach, remaining coefficients can be obtained.
Consider ${p_i}^2 | (p-1)$. Let

$$a_1 \equiv a g^{-x_0} \equiv g^{p_i(x_1 + x_2 p_i + \ ...\ )} (\text{mod } p)$$

Raising both sides to the power of $\frac{(p-1)}{{p_i}^2}$ we obtain,

$$\begin{aligned} {a_1}^{\frac{(p-1)}{p_i^2}} &\equiv g^{\frac{p_i(p-1)(x_1 + x_2 p_i + \ ...\ )}{p_i^2}} \\ &\equiv g^{\frac{x_1(p-1)}{p_i}} \cdot \left(g^{p-1}\right)^{(x_2 + x_3 \cdot p_i + ...)} \\ &\equiv g^{\frac{x_1(p-1)}{p_i}} \cdot 1 \ \ [\text{Applying Fermat's little theorem}] \\ \text{Hence, } {a_1}^{\frac{(p-1)}{p_i^2}} &\equiv g^{\frac{x_1(p-1)}{p_i}} \ \ (\text{mod } p) \end{aligned}$$

The value of x_1 can be found by trial and error method. We need to find a such k where $0 \leq k \leq p_i - 1$ for which

$${a_1}^{\frac{(p-1)}{p_i^2}} \equiv g^{\frac{k(p-1)}{p_i}}$$

is satisfied and then $x_1 = k$.

Accordingly, consider ${p_i}^3 | (p-1)$ then $a_2 \equiv a_1 \cdot g^{-x_1 \cdot p_i}$ and raising the both sides of the power of $\frac{(p-1)}{p_i^3}$, we will obtain x_2.

Following the same approach we will determine all x_j and after determining $x_0, x_1, \ldots, x_{n-1}$, we can calculate x mod p_i. We repeat the above process for all prime factors of $p-1$. Then we will obtain x (mod ${p_i}^{n_i}$) for all i.

Then with the help of Chinese Remainder Theorem, we combine those results or congruences and obtain solution for x (mod $p-1$). Here, $0 \leq x < p-1$ and hence the expected solution x will be found.

3.3.4.3 Time and space complexities for Silver-Pohlig-Hellman algorithm

Pohlig and Hellman represented that if

$$p-1=\prod_{i=1}^{k} {p_i}^{n_i}$$

where p_i are distinct primes, $n_i \geq 1$, $0 \leq r_i \leq 1$, $i \in \{1,2,\ldots,k\}$.

Logarithms over $GF_{(p)}$ can be computed in $\mathscr{O}(\sum_{i=1}^{k} n_i[log_2\ p + {p_i}^{1-r_i}\ (1+log_2\ {p_i}^{r_i})])$ operations using $\mathscr{O}log_2\ p\sum_{i=1}^{k}(1+{p_i}^{r_i})$ bits of memory.

Precomputation needs $\mathscr{O}(\sum_{i=1}^{k}({p_i}^{r_i} log_2\ {p_i}^{r_i} + log_2\ p))$ operations.

If all the prime factors of $p-1$ are small, this algorithm is very efficient.

Example 3.12 Let us consider $3^x \equiv 2 \pmod{617}$ using Silver Pohlig Hellman Algorithm.

Here, prime, $p=617$ and primitive root, $g=3$. Prime factorization of $\phi(p)$ is

$$\phi(p)=\phi(617)=616=2^3 \cdot 7 \cdot 11$$

For $p_1=2$, we need to find $x \pmod{2^3}$. We can write ,

$$x \equiv x_0+2x_1+4x_2 \pmod 8.$$

Let $a \equiv g^x$.

Raising both sides to the power of $\left(\frac{p-1}{p_i}\right)$ we get,

$$a^{\frac{(p-1)}{p_1}} \equiv g^{\frac{x_0(p-1)}{p_1}} \pmod p$$
$$a^{\frac{(p-1)}{2}} \equiv g^{\frac{x_0(p-1)}{2}} \pmod p$$

$$\text{Now, } a^{\frac{p-1}{2}} \equiv 2^{308} \equiv 1 \pmod{617}$$
$$\left(g^{\frac{(p-1)}{2}}\right)^{x_0} \equiv \left(3^{308}\right)^{x_0} \equiv (616)^{x_0} \pmod{617}$$

Now, we need to determine x_0 by trial and error method. However, it can be easily understood that $x_0=0$.

Here, $p_1^2 \mid p-1$;

$$a_1 \equiv a \cdot g^{-x_0} \equiv 2 \cdot 3^{-0} \pmod{p} = 2 \pmod{617}.$$

$$\text{Let } a_1 \equiv a \cdot g^{-x_0} \equiv g^{p_1(x_1+x_2p_1+\dots)} \pmod{p}.$$

Raising both sides to the power of $\frac{(p-1)}{p_1^2}$ we get,

$$a_1^{\frac{(p-1)}{p_1^2}} \equiv g^{\frac{x_1(p-1)}{p_1}} \pmod{p}.$$

Now,

$$a_1^{\frac{(p-1)}{p_1^2}} \equiv (2)^{\frac{616}{4}} \equiv 2^{154} \equiv 1 \pmod{617}$$

and

$$g^{\frac{x_1(p-1)}{p_1}} \equiv \left(3^{308}\right)^{x_1} \equiv (616)^{x_1} \pmod{617}.$$

Therefore, $x_1 = 0$. Continuing the same way,

$$a_2 \equiv a_1 \cdot g^{-x_1p_1} \equiv 2 \cdot 3^0 \pmod{617} \equiv 2 \pmod{617}.$$

Raising both sides to the power of $\frac{(p-1)}{p_1^3}$ we get,

$$a_2^{\frac{(p-1)}{p_1^3}} \equiv \left(g^{\frac{(p-1)}{2}}\right)^{x_2} \pmod{p}$$

$$a_2^{\frac{(p-1)}{p_1^3}} \equiv 2^{\frac{616}{8}} \equiv 2^{77} \equiv 616 \pmod{617}$$

$$\text{and, } \left(g^{\frac{(p-1)}{2}}\right)^{x_2} \equiv \left(3^{308}\right)^{x_2} \equiv (616)^{x_2} \pmod{617}.$$

Therefore, x_2 = 1. Hence,

$$\begin{aligned} & x \equiv x_0 + 2x_1 + 4x_2 \pmod{8} \\ \Rightarrow\ & x \equiv 0 + 2 \cdot 0 + 4 \cdot 1 \equiv 4 \pmod{8} \\ \Rightarrow\ & x \equiv 4 \pmod{8}. \end{aligned}$$

Further, consider $p_i = p_2 = 7$ to determine $x \pmod{7}$. We have,

$$a^{\frac{p-1}{7}} \equiv \left(g^{\frac{(p-1)}{7}}\right)^{y_0} \pmod{617}$$

$$a^{\frac{(p-1)}{7}} \equiv 2^{88} \equiv 420 \pmod{617}$$

$$\text{and } \left(g^{\frac{(p-1)}{7}}\right)^{y_0} \equiv \left(3^{88}\right)^{y_0} \equiv (420)^{y_0} \pmod{617}.$$

So $y_0 = 1$. Hence, $x \equiv 1 \pmod{7}$.

Further $P_i = P_3 = 11$ to determine $x \pmod{11}$. We have,

$$a^{\frac{(p-1)}{11}} \equiv \left(g^{\frac{(p-1)}{11}}\right)^{z_0} \pmod{617}$$
$$a^{\frac{(p-1)}{11}} \equiv 2^{56} \equiv 418 \pmod{617}$$
$$\text{and } \left(g^{\frac{(p-1)}{11}}\right)^{z_0} \equiv \left(3^{56}\right)^{z_0} \equiv (31)^{z_0} \pmod{617}.$$

Applying trial and error method,

$(31)^0 \equiv 0$; $(31)^1 \equiv 31$; $(31)^2 \equiv 344$;
$(31)^3 \equiv 175$; $(31)^4 \equiv 489$; $(31)^5 \equiv 351$;
$(31)^6 \equiv 392$; $(31)^7 \equiv 429$; $(31)^8 \equiv 342$;
$(31)^9 \equiv 113$; $(31)^{10} \equiv 418$;

Hence, $z_0 = 10$, since $(31)^{10} \equiv 418 \pmod{617}$.

So $x \equiv 10 \pmod{11}$. Now, we need to solve the following system of simultaneous congruences by Chinese remainder theorem,

$$x \equiv 4 \pmod{8}$$
$$x \equiv 1 \pmod{7}$$
$$x \equiv 10 \pmod{11}$$

Consider, $a_1 = 4$, $a_2 = 1$, $a_3 = 10$, $n_1 = 8$, $n_2 = 7$, $n_3 = 11$, $N = 8 \cdot 7 \cdot 11 = 616$.

$$\begin{aligned} x &\equiv \sum a_i \cdot \frac{N}{n_i}\left[\left(\frac{N}{n_i}\right)^{-1}\right]_{n_i} \pmod{N} \\ &\equiv 4 \cdot 77[(77)^{-1}]_8 + 1 \cdot 88[(88)^{-1}]_7 + 10 \cdot 56[(56)^{-1}]_{11} \\ &\equiv 4 \cdot 77 \cdot 5 + 1 \cdot 88 \cdot 2 + 10 \cdot 56 \cdot 1 \\ &\equiv 2276 \pmod{616} \\ &\equiv 428 \pmod{616}. \end{aligned}$$

Hence the value of x is 428 which solves $428 \equiv \log_3 2 \pmod{617}$.

Hence,

$$3^{428} \equiv 2 \pmod{617}.$$

Notes on unification of $\mathbb{Z}_p^*$ and $E(\mathbb{F}_p)$

The multiplicative group of a finite field and the group of points on an elliptic curve over a finite field are two of the most prominent examples of finite abelian groups used in public-key cryptography. The hardness of solving the discrete logarithm problem in the finite group $E(\mathbb{F}_p)$ of points of an elliptic curve E over a finite field $\mathbb{F}_p$ is the basis for elliptic curve cryptography. One of the significant characteristics of the DLP is that it may be defined over any cyclic groups and is not limited to the multiplicative group $\mathbb{Z}_p^*$. Elliptic curve discrete logarithm problem is the generalization of DLP which extends the multiplicative group $\mathbb{F}_p^*$ to the elliptic curve group $E(\mathbb{F}_p)$. Considering the cryptographic significance, DLP in $\mathbb{Z}_p^*$ and DLP in $E(\mathbb{F}_q)$ have been discussed in this paper.

We already know that the considered multiplicative group $\mathbb{Z}_p^*$ in the previous sections is an abelian group and when p is a prime the group is a field. In the next chapter, elliptic curve cryptography will be discussed. It will be observed that the elliptic curve is considered over the field $\mathbb{K}$ under the binary operation: 'addition'. To be noted elements on E are closed, associative, and commutative, and it has an identity (the point at infinity) and own inverse under the addition. Therefore, $(E, +)$ forms an abelian group.

Here, we would like to mention that though the DLP over the group of integers under multiplication and DLP over the elliptic curves have been discussed separately in our paper, the discrete logarithm problem can be introduced generally for arbitrary abelian groups. Hence, both concepts can be unified.

4 Elliptic Curve, Elliptic Curve Discrete Logarithm: Application and Attack

The proposal of using elliptic curve discrete logarithm problem (ECDLP) for constructing cryptographic systems was given by Miller [16] and Koblitz [12]. More information can be found on elliptic curve and ECDLP based cryptography can be found in [3].

The same level of security as integer factorization and discrete logarithm system can be obtained in elliptic curve cryptosystem with significantly less operands [17] [4]. Elliptic curve cryptography is set up on the generalized DLP and the security of elliptic curve cryptosystem and elliptic curve digital signature algorithm are based on the hardness of elliptic curve discrete logarithm problem.

4.1 Topics on Elliptic Curve

Definition 4.1 An *elliptic curve* over a field $\mathbb{K}$ is a curve or a set of points that satisfies an equation of the form *(Weierstrass equation)*:

$$y^2 = x^3 + ax + b \text{ where, } a, b \in \mathbb{K} \text{ and}$$

the discriminant of the equation $-16(4a^3 + 27b^2) \neq 0$ or $(4a^3 + 27b^2) \neq 0$.

A *point at infinity* is also considered that is denoted by ∞.

Here, $-16(4a^3 + 27b^2) \neq 0$ represents that the curve is non singular that means all roots are distinct. In most of the cases, equations of the form $y^2 = x^3 + ax + b$ is sufficient for elliptic curves. In few cases, it is required to consider elliptic curves with the equation of the form

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

Nevertheless, in our studies the above expression is not required.

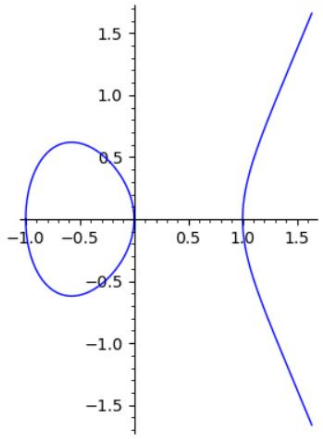

Figure 4.1: Figure for $y^2 = x^3 - x$ over $\mathbb{R}$

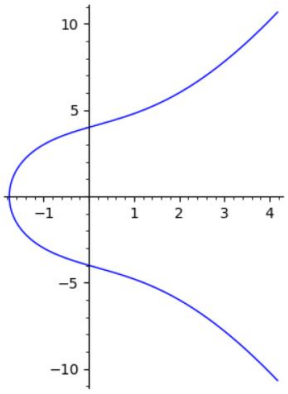

Figure 4.2: Figure for $y^2 = x^3 + 6x + 16$ over $\mathbb{R}$

4.1.1 Properties of Group Operation for Elliptic Curves

There are a few properties of group operation for elliptic curve that can be found in [17].

Let the equation of the elliptic curve E is given by $y^2 = x^3 + ax + b$ and two points on E are $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$.

Then,

$$P_1 + P_2 = R;\ R = (x_r, y_r) \text{ where}$$

$$x_r = m^2 - x_1 - x_2;\ y_r = m(x_1 - x_r) - y_1 \text{ and}$$

$$m = \begin{cases} \frac{(y_2 - y_1)}{(x_2 - x_1)} & \text{if } P_1 \neq P_2 \\ \\ \frac{(3x_1^2 + a)}{(2y_1)} & \text{if } P_1 = P_2 \end{cases}$$

To illustrate, m is the *slope* of the line passing through P_1 and P_2 where P_1 and P_2 are distinct points or m is the slope of the tangent that passes through P_1 (in case of both points being same).

One *additional law* is $P + \infty = P$ for all points P on the elliptic curve. This point at infinity (∞) is defined as the identity element of this group. It is considered that 'point at infinity' is located along the top and bottom of the y-axis.

According to the definition, *inverse of any element* $P(x, y)$ of the group is defined as $-P(x, -y)$ [13] (i.e. simply multiplying its Y coordinate by -1) and can be written that $P + (-P) = \infty$.

Remark 4.2 When elliptic curve is considered over the prime field $G(\mathbb{F}_p)$, inverse of an element can be obtained simply since -$y_p \equiv p - y_p \pmod p$. Hence, $-P = (x_p, p - y_p)$.

4.1.1.1 Addition of two points on elliptic curve

Let $P_1(x_1,y_1)$ and $P_2(x_2,y_2)$ be two distinct points on E.

We obtain another point R on E such that $P_1 + P_2 = R$. We draw a line L that passes through P_1 and P_2 and this line L intersects the curve E in third point Q.

Then, we consider and reflect the point Q through the x-axis (change y to $-y$, i.e. multiplying its Y coordinate by -1) and obtain another point R.

This point R is expressed as

$$P_1 + P_2 = R.$$

It should be noted that the operation used above is not as same as adding points in the plane. If P_1, P_2 are not distinct i.e. $P_1 = P_2$, then we need to consider point doubling.

Example 4.3 Let an elliptic curve E given by $y^2 = x^3 + 6x + 16$. Consider, $P_1 = (2,6)$ and $P_2 = (8,24)$ are two points on E.

The equation of the straight line that passes through P_1 and P_2 is $y = 3x$.

Substituting the value of y into the equation of elliptic curve, we obtain,

$$\begin{aligned} (3x)^2 &= x^3 + 6x + 16 \\ \Rightarrow x^3 - 9x^2 + 6x + 16 &= 0 \end{aligned}$$

The line intersects the curve in P_1 and P_2 and a third point Q. Two of the roots (out of three) are $x = 2$ and $x = 8$. Now, Q is needed to be determined to determine R.

We know that sum of three roots is minus the coefficient of x^2 divided by coefficient of x^3 and in our case, sum of the roots is 9.

Hence, for the third point $x_3 = -1$.

Since $y = 3x$; $y = 3(-1) = -3$.

So, $Q = (-1,-3)$. Hence, the result is $R = (-1,3)$.

Addition of two points $P_1 = (2,6)$ and $P_2 = (8,24)$ on $E : y^2 = x^3 + 6x + 16$ can be visualized in the figure 4.3.

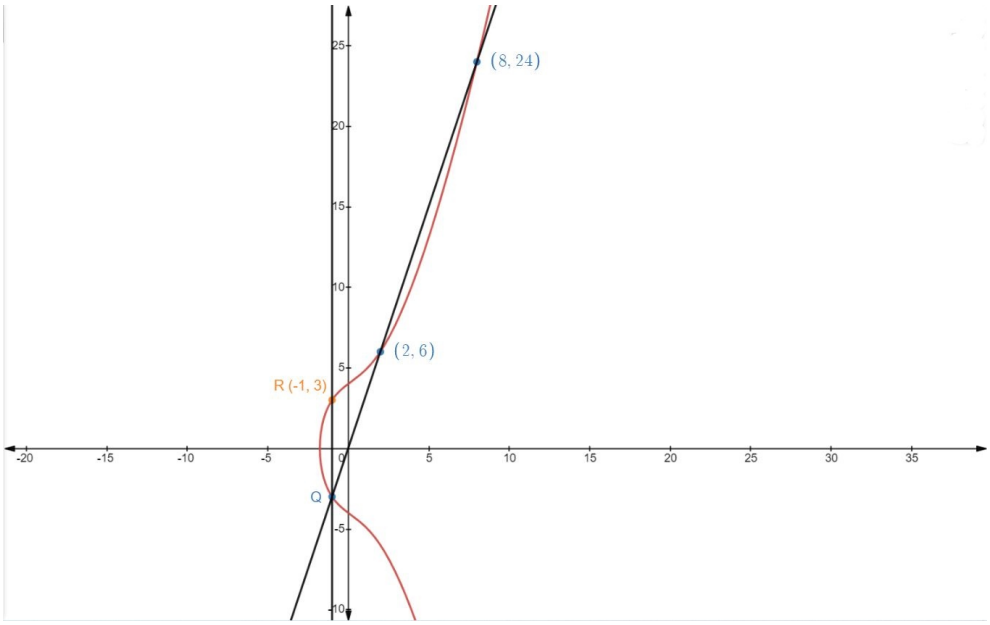

Figure 4.3: Addition of two points on Elliptic curve:

4.1.1.2 Point doubling on elliptic curve

When both point are same i.e. $P_1 = P_2$, we can write $R = P_1 + P_1 = 2P_1$. To obtain R we need a slightly different approach here. A tangent line is drawn through P_1. Then, a second point of intersection between this line and the elliptic curve is obtained. We consider the reflection of the second intersection along the x-axis and we obtain a third point R on E.

4.1.1.3 Adding the point P_1 and point at infinity (∞) for elliptic curve

A line that passes through P and ∞ is vertical and therefore the points of intersections between the line and E are $P = (x, y)$ and $(x, -y)$. After considering the reflection of $(x, -y)$ across the x-axis, we obtain $R = (x, y) = P$, Hence, $P + \infty = P$.

4.1.1.4 Subtracting the point on elliptic curve

The line that passes through (x, y) and $(x, -y)$ is vertical and therefore the third point of intersection between the vertical line and E is at ∞. Then, we consider the reflection across the x-axis which is also at ∞. Hence, $(x, y) + (x, -y) = \infty$ for which it can defined that $(x, -y) = -(x, y)$. [∞ being the additive identity]. Therefore, if it requires to subtract P_2 from P_1, then the result is addition of P_1 and (-P_2).

4.1.2 Illustration on operations for elliptic curve

If $P_1 \neq P_2$, $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$ are two distinct points. We also consider $P_1 \neq -P_2$. Then slope of the line passing through P_1 and P_2 is

$$m = \frac{(y_1 - y_2)}{(x_1 - x_2)}$$

and the equation of the line passing through P_1 and P_2 is $y = mx + c$.

Since $P_1(x_1, y_1)$ is a point that lies on the line, $y = mx + c$; we can write,

$$y_1 = mx_1 + c \Rightarrow \ c = y_1 - mx_1.$$

By substituting c value in the equation $y = mx + c$, we get,

$$y = mx + y_1 - mx_1 \Rightarrow \ y = m(x - x_1) + y_1$$

By substituting the above term of y in the equation $y^2 = x^3 + ax + b$, the third intersection point can be found:

$$m(x - x_1) + y_1)^2 = x^3 + ax + b$$

$$\begin{aligned}
&\Rightarrow m^2(x^2 - 2xx_1 + {x_1}^2) + 2my_1(x - x_1) + {y_1}^2 = x^3 + ax + b \\
&\Rightarrow m^2x^2 - 2m^2xx_1 + m^2{x_1}^2 + 2mxy_1 - 2mx_1y_1 + {y_1}^2 = x^3 + ax + b \\
&\Rightarrow x^3 - m^2x^2 + ax + 2m^2xx_1 - 2mxy_1 + b - m^2{x_1}^2 + 2mx_1y_1 - y_1^2 = 0 \\
&\Rightarrow x^3 - m^2x^2 + (a + 2m^2x_1 - 2my_1)x + (b - m^2x_1^2 + 2mx_1y_1 - y_1^2) = 0
\end{aligned}$$

It is known that sum of the three roots of a cubic equation is the minus of coefficient of x^2 divided by coefficient of x^3. We know (x_1, y_1) and (x_2, y_2) are the intersection point of the line and the curve. Let the coordinates of third intersection point is $(x_{r'}, y_{r'})$.

Therefore,

$$x_1 + x_2 + x_{r'} = m^2 \Rightarrow x_{r'} = m^2 - x_1 - x_2.$$

We have,

$$y = m(x - x_1) + y_1.$$

Replacing x and y by $x_{r'}$ and $y_{r'}$ respectively, we obtain,

$$y_{r'} = m(x_{r'} - x_1) + y_1.$$

If we take the third point $x_{r'}, y_{r'}$ and reflect through x-axis, another point R intersects the curve and the new coordinates will be $R(x_r, y_r)$; where, $x_r = x_{r'}$, $y_r = -y_{r'}$.

So,

$$R = (m^2 - x_1 - x_2,\ m(x_1 - x_r) - y_1).$$

Hence,

$$P_1 + P_2 = R.$$

Now, we will consider the case when P_1 and P_2 are same point that means $P_1 = P_2 = (x_1, y_1)$. If we consider the differentiation of the expression of elliptic curve: $y^2 = x^3 + ax + b$, to obtain slope, we get,

$$\frac{dy}{dx} = \frac{3x^2 + a}{2y}.$$

So, at point P_1, the slope is

$$m = \frac{3x_1^2 + a}{2y_1}.$$

So, at point $P_1(x_1, y_1)$, the equation of tangent line with slope m is

$$y - y_1 = m(x - x_1) \Rightarrow y = m(x - x_1) + y_1 \text{ where } m = \frac{3x_1^2 + a}{2y_1}.$$

Continuing the same approach done earlier, we obtain,

$$x^3 - m^2x^2 + (a + 2m^2x_1 - 2my_1)x + (b - m^2x_1^2 + 2mx_1y_1 - y_1^2) = 0.$$

Since, $P_1 = P_2$, we consider $(x_{r'}, y_{r'})$ intersects the curve as third point. So,

$$x_1 + x_1 + x_{r'} = m^2 \Rightarrow x_{r'} = m^2 - 2x_1.$$

Further, we get,

$$y_{r'} = m(x_{r'} - x_1) + y_1.$$

If we take the third point and reflect through x axis, another new point $R(x_r, y_r)$ intersects the curve and

$$R = (m^2 - 2x_1, m(x_1 - x_r) - y_1);\ \text{where } m = \frac{3x_1^2 + a}{2y_1}.$$

4.1.2.1 Elliptic curves mod n

Earlier, we studied elliptic curves geometrically. Now, we want to apply the theories of elliptic curves in the field of cryptography and to do so we need to consider elliptic curves having coordinates in a finite field $\mathbb{F}_{(p)}$.

For elliptic curves mod n, addition of points is computed in the same way using same laws, discussed earlier, but there is an exception. For any rational number $\frac{a}{b}$ is needed to be considered ab^{-1} where, $b^{-1}b$ = (mod n) if gcd$(b, n) = 1$.

Example 4.4 We consider an elliptic curve $E : y^2 = x^3 + 6x + 4$ over the field $\mathbb{F}_{11}$. The points of $E(\mathbb{F}_{11})$ can be found by substituting the possible values for x modulo 11 (i.e. $0, 1, \ldots\ldots, 10$) and consider the values for which the given expression is satisfied : ($x^3 + 6x + 4$ is a square modulo 11).

Considering the substitution for x, we get,

$x \equiv 0$	$y^2 \equiv 4 \pmod{11}$	$y \equiv 2, 9 \pmod{11}$
$x \equiv 1$	$y^2 \equiv 11 \equiv 0 \pmod{11}$	$y \equiv 0$
$x \equiv 2$	$y^2 \equiv 24 \equiv 2 \pmod{11}$	No solution
$x \equiv 3$	$y^2 \equiv 49 \equiv 5 \pmod{11}$	$y \equiv 4, 7$
$x \equiv 4$	$y^2 \equiv 92 \equiv 4 \pmod{11}$	$y \equiv 2, 9$
$x \equiv 5$	$y^2 \equiv 159 \equiv 5 \pmod{11}$	$y \equiv 4, 7$
$x \equiv 6$	$y^2 \equiv 256 \equiv 3 \pmod{11}$	$y \equiv 5, 6$
$x \equiv 7$	$y^2 \equiv 389 \equiv 4 \pmod{11}$	$y \equiv 2, 9$
$x \equiv 8$	$y^2 \equiv 564 \equiv 3 \pmod{11}$	$y \equiv 5, 6$
$x \equiv 9$	$y^2 \equiv 787 \equiv 6 \pmod{11}$	No solution
$x \equiv 10$	$y^2 \equiv 1064 \equiv 8 \pmod{11}$	No solution

Table 4.1: Finding points on $E : y^2 = x^3 + 6x + 4$ over the field F_{11}.

So, points on E are : $(0,2), (0,9), (1,0), (3,4), (3,7), (4,2), (4,9), (5,4), (5,7), (6,5), (6,6), (7,2), (7,9), (8,5), (8,6), (\infty, \infty)$.

We consider the same elliptic curve to add two points $P_1(4,2)$ and $P_2(8,5)$ on the curve. Since two points are given, slope,

$$m = \frac{5-2}{8-4} = \frac{3}{4}.$$

Hence,

$$x_3 \equiv (\frac{3}{4})^2 - 4 - 8 \equiv \frac{-183}{16} \pmod{11} \equiv -183 \cdot 9 \equiv -1647 \equiv 3 \pmod{11};$$
$$y_3 \equiv \frac{3}{4}(4-3) - 2 \pmod{11} \equiv \frac{-5}{4} \pmod{11} \equiv -5 \cdot 3 \equiv 7 \pmod{11}.$$

So, $P_1 + P_2 = (3,7)$.

We are interested to consider another example where we want to add a point with itself.

Example 4.5 Let $E : y^2 \equiv x^3 + 6x + 16 \pmod{1259}$. Consider, $P_1 = (2,6)$. According to the point doubling, $P_1 + P_1 = 2P_1$.

We consider, the differentiation of the expression of elliptic curve to obtain slope and therefore,

$$\frac{dy}{dx} = \frac{(3x^2+6)}{2y}$$
$$\text{At } P_1, \quad \frac{dy}{dx} = \frac{(3\cdot 2^2+6)}{2\cdot 6} = \frac{3}{2}.$$

As we are working with modulo and considering extended euclidean algorithm, we can write,

$$m \equiv \frac{3}{2} \pmod{1259} \equiv 3\cdot 630 \equiv 631 \pmod{1259}.$$

Using the formula we have,

$$x_3 = m^2 - x_1 - x_2 \pmod{1259};$$
$$y_3 = m(x_1 - x_3) - y_1 \pmod{1259}.$$

$$\text{So,} \quad x_3 = (631)^2 - 2 - 2 \equiv 313 \pmod{1259};$$
$$\text{and} \quad y_3 = 631(2-313) - 6 \equiv 157 \pmod{1259}.$$

Hence, $2P_1 = P_1 + P_1 = (313, 157)$.

The figure 4.4 represents all possible points for the elliptic curve $E : y^2 \equiv x^3 + 6x + 16 \pmod{1259}$ used in above example.

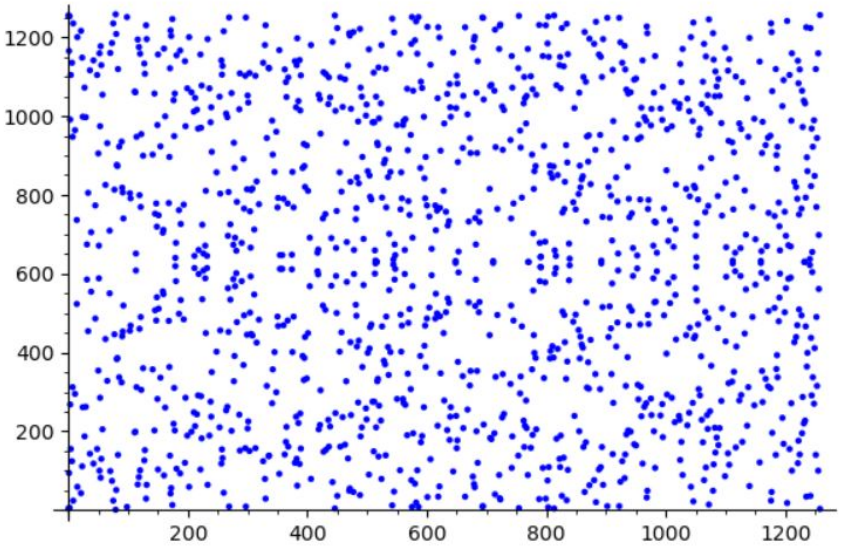

Figure 4.4: Figure for $y^2 \equiv x^3 + 6x + 16 \pmod{1259}$:

4.2 Elliptic Curve Discrete Logarithm Problem

Definition 4.6 Consider E be an elliptic curve over a finite field $\mathbb{F}_q$. Let P, R are points on elliptic curve and

$$\underbrace{P+P+\cdots+P}_{\text{n times}} = nP = R \text{ for some integer } n$$

The problem of finding n is termed as elliptic curve discrete logarithm problem, ECDLP. Here, $R \in < P >$; we write $R = nP$ where integer $n \in [0, q-1]$; q is the order of P and n is called the discrete logarithm of R to the base P, which means $n = log_P R$. [8]

In cryptosystem, the integer n is considered as the private key and the point $R(x_r, y_r)$ is considered as public key. On the other hand, for classical discrete logarithm problem (DLP) in $\mathbb{Z}_P{}^*$ both of public key and private key were integers.

The operation in the expression $P+P+\cdots+P = nP = R$ is termed as point multiplication. Though the notation np looks similar as arithmetic operation or multiplication, indeed it is not. It is only a notation for the convenience and used for repeated application of group operation. In order to break cryptosystem, the private key n is needed to be figured out or revealed.

Example 4.7 Consider $y^2 = x^3 + 6x + 4 \pmod{11}$. Let $P = (3,4)$ be a primitive element under addition as group operation.
We compute $P,\ P+P = 2P,\ P+2P = 3P,\ \ldots$ as follows:

$P = (3,4);$	$2P = (5,7);$	$3P = (8,5);$	$4P = (4,9);$
$5P = (7,9);$	$6P = (6,6);$	$7P = (0,9);$	$8P = (1,0);$
$9P = (0,2);$	$10P = (6,5);$	$11P = (7,2);$	$12P = (4,2);$
$13P = (8,6);$	$14P = (5,4);$	$15P = (3,7);$	$16P = \infty;$

Then the elements are repeated under operation. $17P = (3,4);\ 18P = (5,7);\ 19P = (8,5);\ 20P = (4,9); \ldots\ldots\ldots$

Here, we have obtained $P = (3,4)$ and $P + 15P = 16P = \infty$ which represents that $15P$ is the inverse of P and vice-versa. Indeed for both point, the X-coordinates are 3 and Y-coordinates are additive inverse of each other for modulo 11 since $4 \equiv 7 \equiv -4 \pmod{11}$. We can have another look on the example: $E : y^2 = x^3 + 6x + 4 \pmod{11}$ to understand point multiplication.

We had $P = (3,4)$ and $10P = \underbrace{P+P+\cdots+P}_{\text{10 times}} = (6,5)$. Here, $n = 10$. Using double and add algorithm np can be computed.

4.3 Elliptic Curve Discrete Logarithm Based Cryptography

An elliptic curve cryptosystem can achieve the same level of security as integer factorization and discrete logarithm systems with significantly less operands. The securities of elliptic curve cryptosystems including elliptic curve digital signature algorithms are dependent on the hardness of the ECDLP. Due to those special properties, elliptic curve cryptography is commonly utilized to design strong cryptosystems and many cryptosystems are built based on intractability of ECDLP.

4.3.1 Diffie Hellman Key Exchange over Elliptic Curve

Now, we can consider elliptic curve over the cryptography. The Diffie Hellman key exchange, a simple but significant method that has been studied earlier, can be extended in terms of elliptic curve [16] [9].

Let Alice and Bob wants to communicate confidentially and want to exchange the key over an unsecured channel. To do so, they publicly agree on a large prime, an elliptic curve E over $\mathbb{F}_p$, a base point G in $E(\mathbb{F}_p)$.

Alice chooses a random number $x \in \{2,3,4,\ldots,\#E-1\}$ and computes $xG = A$.
She must keep x secret. Then Alice sends A to Bob.

Bob chooses a random number $y \in \{2,3,4,\ldots,\#E-1\}$ and keeps y secret. Bob computes $yG = B$ and sends B to Alice. After that, Alice calculates

$$B^x = xB = x(yG) = xyG.$$

Bob calculates

$$A^y = yA = y(xG) = xyG.$$

Since point addition is associative, both of Alice and Bob actually computes the same point xyG.

Example 4.8 Let Alice and Bob agree to use elliptic curve Diffie Hellman with $p = 2579$. $E : y^2 = x^3 + 12x - 237 \pmod{2579}$, $G = (285, 2554) \in E(\mathbb{F}_{2579})$.

Alice chooses a random integer $x = 1126$ and keeps secret.
She computes,

$$A = 1126\ G = 1126\ (285, 2554)\ = (518, 259) \in E(\mathbb{F}_{2579}).$$

Alice sends A to Bob.

Bob chooses a random integer $y = 1245$ and keeps secret.
Bob computes,

$$B = yG = 1245(285, 2554) = (822, 2394) \in E(\mathbb{F}_{2579}).$$

Bob sends B to Alice.

Now, Alice calculates,

$$xB = 1126(822, 2394) = (2468, 2011) \in E(\mathbb{F}_{2579}).$$

Further, Bob calculates,

$$yA = 1245(518, 259) = (2468, 2011) \in E(\mathbb{F}_{2579}).$$

So, Alice and Bob have shared the secret point $(2468, 2011)$. Indeed, Alice and Bob both discard the y coordinate of the point and consider the value of x coordinate as the secret shared key which is $x = 2468$.

4.3.2 Elliptic Curve ElGamal Cryptosystem

We considered the non-elliptic curve (classical) version of ElGamal cryptosystem earlier which also has elliptic curve analog. Now we are to discuss about elliptic curve version of the method.

Bob needs to choose an elliptic curve $E \pmod p$ where p is a large prime.
Further, he chooses a base point (primitive point) G on E. He also selects an integer y and keeps secret.Then, he computes,

$$B = yG = \underbrace{G + G + \cdots + G}_{\text{y times}}$$

He declares p, G, B public along with E.

For sending the message M as a point on the curve E. Alice chooses a random integer x and computes,

$$C_1 = xG \text{ and } C_2 = M + xB.$$

Then, Alice sends (C_1, C_2) to Bob.

To recover the message M, Bob computes,

$$M = C_2 - y\,C_1.$$

We can have a look why the process works:

$$\begin{aligned} C_2 - yC_1 &= M + xB - yxG \\ &= M + (xy)G - (xy)G \\ &= M. \end{aligned}$$

Though theoretically, elliptic curve ElGamal cryptosystem suits well, it faces difficulties in terms of attaching plain text massage to points on E($\mathbb{F}_p$). Approaches to overcome this issue can be found in [23].

Example 4.9 Let Alice and Bob both agree to use $E : y^2 = x^3 + 5x + 89 \pmod{5683}$ where, 5683 is a prime.

Bob chooses a base point $G = (135, 20)$.

He selects a secret integer $y = 106$.

Now, Bob computes,

$$B = yG = 106(135, 20) = (1746, 2547).$$

Then, he makes G and B public also.

So, the public information is (p, a, b, G, B) i.e, $(5683, 5, 89, (135, 20), (1746, 2547))$.

Let Alice wants to send $(73, 443)$ as message M.

Now, Alice downloads the information. She chooses secret integer $x = 147$.

Then, she calculates,

$$C_1 = xG = 147(135, 20) = (2251, 4961)$$

she also calculates,

$$\begin{aligned} C_2 &= M + 147(1746, 2547) \\ &= (73, 443) + (865, 5278) \\ &= (1604, 450). \end{aligned}$$

Alice sends (C_1,C_2) to Bob.

After receiving Bob calculates,

$$y\,C_1 = 106(2251, 4961) = (865, 5278)$$

Then, he subtracts $y\,C_1$ from C_2, i.e, He computes,

$$\begin{aligned} C_2 - y\,C_1 &= (1604, 450) - (865, 5278) \\ &= (1604, 450) + (865, -5278) \\ &= (73, 443) \end{aligned}$$

Therefore, he recovers M.

4.3.3 Elliptic Curve Digital Signature Algorithm

It is also possible to represent digital signature schemes in terms of elliptic curve over $\mathbb{F}_p$ with order q as like as other cryptosystems. Out of several digital signature schemes, one popular varaint of ElGamal's signature scheme for elliptic curve analogue named as ECDSA will be described here . [10] [9]

Both of Alice and Bob agree on a finite field $\mathbb{F}_p$ where p is a large prime, an elliptic curve over $\mathbb{F}_p$ and a base point $G \in E(\mathbb{F}_p)$ with order q.

Alice chooses a random secret integer x and computes $A = xG$. She publishes A so that the information on (p, a, b, q, G, A) is publicly available.

Alice chooses a session key k such that $1 \leq k \leq q-1$; gcd $(k, q) = 1$ and computes $T = kG = (x_t, y_t)$. She also computes

$$u = k^{-1}(m + x\,x_t) \pmod{q}.$$

Alice sends the signed message (m, T, u).

To verify Bob computes,

$$\begin{aligned} v_1 &= m \cdot u^{-1} \pmod{q}; \\ v_2 &= x_t \cdot u^{-1} \pmod{q}. \end{aligned}$$

Then, he computes

$$V = v_1 G + v_2 A.$$

Bob accepts the signature if $V = T$ since

$$G^{v_1} \cdot A^{v_2} = G^{mu^{-1}} \cdot (G^x)^{x_t\ u^{-1}} = G^{(m+x\ x_t)u^{-1}} = G^k = kG = T.$$

Example 4.10 Alice wants to send a signed message to Bob using ECDSA.

Alice chooses an elliptic curve with $a = 15$, $b = 101$ and prime, $p = 3571$; i.e, $E : y^2 = x^3 + 15x + 101 \pmod{3571}$. She also chooses the base point $G = (125, 42)$ with order $q = 3577$. Then she chooses a secret integer $x = 110$ and computes

$$A = xG = 110(125, 42) = (2633, 993)$$

Then, the information (p, a, b, q, G, A) is made public i.e,

$$(3571, 15, 101, 3577, (125, 42), (2633, 993)).$$

Now, consider Alice wants to send hashed message $m = 234$. She chooses a session key $k = 137$ and computes

$$T = kG = 137(125, 42) = (1730, 3558) = (x_t, y_t).$$

Consider $x_t = 1730$ and she also computes

$$\begin{aligned} u &= k^{-1}(m + x \cdot x_t) \pmod{q} \\ &= 1906 \cdot (234 + 110 \cdot 1730) \pmod{3577} \\ &= 2879 \pmod{3577}. \end{aligned}$$

Alice send (m, T, u). To verify Bob computes

$$v_1 = m \cdot u^{-1} \pmod{q} = 234 \cdot 1768 \pmod{3577} = 2357 \pmod{3577}.$$

He also calculates

$$v_2 = x_t \cdot u^{-1} \pmod{q} = 1730 \cdot 1768 \pmod{3577} = 305 \pmod{3577}.$$

Then, Bob computes

$$\begin{aligned} V &= v_1 G + v_2 A \\ &= 2357(125, 42) + 305(2633, 993) \\ &= (1591, 3436) + (2808, 2805) \\ &= (1730, 3558). \end{aligned}$$

Here, $V = T$. Therefore, Bob accepts the signature.

4.4 Classical Algorithms for Elliptic Curve Discrete Logarithms

Algorithms for elliptic curve discrete logarithms can be divided into two types: those that work in a general environment regardless of the properties of elliptic curve, and those that work with specific properties of elliptic curve to construct a distinct method. Baby-step Giant-step method and Silver Pohlig Hellman method for ECDLP are examples of the former type. There are also several methods to solve ECDLP.

4.4.1 Baby-Step Giant-Step Method for ECDLP:

Baby-step Giant-step method for discrete logarithm problem can be extended for ECDLP simply. [25]

Consider a prime p , an elliptic curve E over the finite field $\mathbb{F}_p$, a base point P (with order q). Also Q is another point on the curve. It is required to find k such that Q=kP. The following steps are taken to find out the value of k.

Step 1: Let m= $\lceil\sqrt{p}\rceil$.
Using Euclid theorem, we write , k = $im+j$ for $i,j \in \{0,1,2,\ldots,m-1\}$. Now,

$$Q = kP = (im+j)P = [im]P + [j]P$$

Therefore,

$$[j]P = Q - [im]P$$

Here, the values of i and j are not known. If the values of i and j can be figured out, then k can be found easily since $k = im+j$.

Step 2: We compute $[j]P$ for all $j = 0,1,2,\ldots,m-1$ and store the results in table.

Step 3: Then we start computing

$$Q - [im]P \text{ for each } j = 0,1,\ldots,m-1$$

For each computation of Giant step: $Q-[im]P$, it is checked if the obtained value exists in the previous table. If the obtained result exists in table already, it can be said that there is a matching we stop computing giant steps.

Step 4: If a match is found in step 3, the values of i and j satisfies $[j]P = Q-[im]P$ as well. We compute $k = im+j$. Hence, k has been found.

Remark 4.11 In the above method, $Q-[im]P$ represents the Giant step and $[j]P$ rep-

resents the Baby step. [If a match is found, then $Q = [im + j]P$]

Example 4.12 Let us consider the following ECDLP problem using BSGS method, that is to find k such that $Q \equiv kP \pmod{41}$ where, E/$\mathbb{F}_{41}$: $y^2 \equiv (x^3 + 2x + 1) \pmod{41}$, $P = (0,1)$ and $Q = (30,40)$

Let, $m = \lceil \sqrt{41} \rceil = 7$. Consider, $k = im + j$; $i, j \in \{0,1,\ldots,6\}$. We need to find k such that

$$Q \equiv kP \equiv (im + j)P \quad \text{or} \quad Q - [im]P = [j]P$$

So, we compute for $[j]P$ and store the results in the following table:

Baby step j	0	1	2	3	4	5	6
$[j]p$	(0,1)	(0,1)	(1,39)	(8,23)	(38,38)	(23,23)	(20,28)

Now, we start computing,

$$Q - [im]P \quad \text{and obtain,}$$
$$\text{For} \quad i = 0, Q - [im]P = (30,40)$$
$$\text{For} \quad i = 1, Q - 7P = (30,1)$$
$$\text{For} \quad i = 2, Q - 14P = (22,19)$$
$$\text{For} \quad i = 3, Q - 21P = (1,39)$$

An expected matching has been found for $i = 3$ and $j = 2$. So, $[2]P = Q - [21]P$. Hence, $Q = 23P$, implies that, $(30,40) = 23\ (0,1)$.

4.4.2 Silver Pohlig Hellman Attack for ECDLP

It is also possible to extend Silver Pohlig Hellman Attack for ECDLP and now the method will be described here. [3]
Let p be a prime, an elliptic curve over the finite field $\mathbb{F}_p$, a base point P with order q. Given Q is another point on the curve. It is required to find x where $Q = xP$. Let the order q is divisible by prime and then the following steps are considered.

Step 1: We consider the prime factorization of q.

$$q = \prod_{i=1}^{k} p_i^{n_i}$$
$$= p_1^{n_1} \cdot p_2^{n_2} \cdot \cdots \cdot p_k^{n_k}$$

Step 2: Starting with p_1, we consider $p_1|q$ and we compute,

$$Q_0 = [\frac{q}{p_1}]Q \quad \text{and} \quad P_0 = [\frac{q}{p_1}]P$$

By using trial and error method, we find an integer x_0 such that

$$Q_0 = x_0\, P_0$$

Step 3: Considering $p_1^{j+1} \mid q$, where, $j = \{1, \ldots, h-1\}$; h is the highest power of p_1 that divides q. We compute,

$$Q_j = \frac{q}{p^{j+1}}(Q - x_0\, p_1^0\, P - x_1\, p_1^1\, P - \cdots - x_{j-1}\, p_1^{j-1}\, P)$$

For each Q_j, using trial and error method, we try to find an integer x_j such that $Q_j = [x_j]p_0$.

Step 4: We apply the following formula after finding $x_0, x_1, \ldots\ldots, x_{n_i-1}$,

$$x = x_0 + x_1 \cdot p_1 + x_2 \cdot {p_1}^2 + \cdots + x_{n_i-1} \cdot {p_1}^{n_i-1} \ (\text{mod}\ p_1^{n_1})$$

and we obtain,

$$x\ (\text{mod}\ p_1^{n_1})$$

Step 5: We repeat steps [2-4] for each prime factors p_i of q and we obtain,

$$x\ (\text{mod}\ p_i^{n_i}) \text{ for all } i \in \{1, \ldots, k\}.$$

Step 6: We apply Chinese remainder theorem,combining all the congruences, to obtain solution x (mod q). Here, $0 \leq x \leq q-1$ and hence the expected solution of x has been obtained.

Example 4.13 Let us consider the following ECDLP problem using Silver Pohlig Hellman method:

$$E : y^2 \equiv x^3 + 1\ (\text{mod}\ 599)$$

$$P = (60, 19) \text{ and } Q = (277, 239)$$

We find k such that $Q = kP \in E(\mathbb{F}_{599})$.

P has order 600 in the group $E(\mathbb{F}_{599})$. Considering prime factorization, $600 = 2^3 \cdot 3 \cdot 5^2$. For $p_1 = 2$, we need to find $x\ (\text{mod}\ 2^3)$ and it can be written that $x \equiv x_0 + 2x_1 + 4x_2$ (mod 8).

Finding modulo 2^3: Let $p_1 = 2$ and $2 \mid 600$,

At first, we multiply both of P and Q by $\frac{600}{2} = 300$,

$$P_{a0} = 300\ P = 300\ (60,19) = (598,0)$$
$$Q_{a0} = 300\ Q = 300\ (277,239) = (0,1).$$

We need to find x_0 such that

$$Q_{a0} = x_0\ P_{a0}$$
$$\Rightarrow (0,1) = x_0\ (598,0).$$

Using trial and error method, it can be easily found that $x_0 = 2$.
Again, $2^2 \mid 600$,

$$\begin{aligned} Q_{a1} &= \frac{600}{4}\ (Q - x_0\ P) \\ &= 150\ ((277,239) - 2\ (60,19)) \\ &= 150\ (35,243) \\ &= (0,1). \end{aligned}$$

We need to find such x_1 for which

$$Q_{a1} = x_1\ P_{a0}$$
$$\Rightarrow (0,1) = 2\ (598,0).$$

So, $x_1 = 2$ which can be found by trial and error method.
Further, $2^3 \mid 600$,

$$\begin{aligned} Q_{a2} &= \frac{600}{2^3}\ (Q - x_0 P - x_1\ 2P) \\ &= 75\ ((35,243) - (340,353)) \\ &= (598,0). \end{aligned}$$

We need to find x_2 such that $Q_{a2} = x_2\ P_{a0}$. Simply it could be understood that $x_2 = 1$.So,

$$\begin{aligned} x &\equiv x_0 + 2x_1 + 4x_2 \pmod 8 \\ &\equiv 2 + 2 \cdot 2 + 4 \cdot 1 \pmod 8 \\ &\equiv 2 \pmod 8. \end{aligned}$$

Solution modulo 3: Let, $p_2 = 3$ and $3 \mid 600$.

At first, we multiply by both of P and Q by $\frac{600}{3} = 200$ to obtain

$$P_{b0} = 200\ P = 200\ (60,19) = (0,1);$$
$$Q_{bo} = 200\ Q = 200\ (277,239) = (0,598).$$

We need to find such x_0 for which

$$Q_{b0} = x_0\, P_{b0};$$
$$(0,598) = x_0\, (0,1).$$

It can be found that $x_0 = 2$. So,

$$x \equiv 2 \pmod 3.$$

Solution modulo 5^2:

Now $p_3 = 5$ and $5 \mid 600$. We need to find x (mod 5^2). At first we need to multiply both of P and Q by $\frac{600}{5} = 120$. Then, we obtain,

$$P_{c0} = 120\, P = 120\, (60,19) = (84,179);$$
$$Q_{c0} = 120\, Q = 120\, (277,239) = (84,179).$$

We need to find x_0 such that $Q_{c0} = x_0\, P_{c0}$. Here $x_0 = 1$ since $(84,179) = 1\, (84,179)$.

Further, $5^2 \mid 600$,

$$\begin{aligned} Q_{c1} &= \frac{600}{25}\, (Q - x_0\, P) \\ &= 24\, ((277,239) - 1\, (60,19)) \\ &= 24\, (130,129) \\ &= (491,465). \end{aligned}$$

We need to find such x_1 for which

$$Q_{c1} = x_1\, P_{c0}$$
$$\Rightarrow (491,465) = x_1\, (84,179).$$

By trial and error method it can easily be found that $x_1 = 3$. So,

$$x \equiv x_0 + 5x_1 \pmod{25} \equiv 1 + 5 \cdot 3 \pmod{25} \equiv 16 \pmod{25}.$$

Therefore, we have,

$$x \equiv 2 \pmod 8$$
$$x \equiv 2 \pmod 3$$
$$x \equiv 16 \pmod{25}$$

After solving the above system of simultaneous congruences by the Chinese Remainder

Theorem, we get the solution $x = 266$. Therefore,

$$Q = 266\ P \in E(\mathbb{F}_{599})$$

which represents.

$$(277, 239) = 266\ (60, 19).$$

5 Conclusion

Though in a few cases, discrete logarithm problems can be computed fast, in general, there is no efficient method for classical computers to compute DLP as of today. It is assumed that computing discrete logarithm problems over a carefully selected group: for example, large prime order subgroups of groups $\mathbb{Z}_p^*$ is extremely hard. On the basis of the hardness of computing DLP, many cryptosystems have been built up. At present, elliptic curve cryptography has become a standard term as it is popularly used to construct strong cryptosystems due to the intractability of ECDLP. Compared to DLP, the same level of security can be achieved in ECC by using smaller keys.

In this paper, several methods to solve discrete logarithm problems have been discussed. The initial approach: Exhaustive search method was intentionally not included since this method does nothing but computes powers of given primitive element until finding the expected value and therefore not efficient. An advantage of the kangaroo method that has been discussed is that the approach uses a little memory. Since the method can be parallelized, it is possible to make the method n times faster using n processors to find a solution for DLP (Van Oorschot and MJ Wiener -1999). The index calculus method has also been demonstrated which is known as a powerful method for computing discrete logarithms. Indeed it is a non-generic sub-exponential time algorithm. In terms of carefully chosen parameters, the heaviest part of this method is to calculate the logarithms of the elements of the considered factor base. The focus was given to the foundations and deterministic algorithms for DLP.

The baby step giant step method and Silver-Pohlig-Hellman algorithm along with their extensions to the elliptic curve analog have been illustrated. Baby step Giant step algorithm is a type of square root method to compute DLP which works on arbitrary groups. However, it becomes infeasible if the order of the group is larger than 10^{40} [25]. On average, a solution can be found after the computation of half of the giant steps. This deterministic approach has $\mathcal{O}(\sqrt{p}\ log\ p)$ and needs space of $\mathcal{O}(\sqrt{p})$. Another deterministic approach, the Silver-Pohlig-Hellman algorithm can be performed if the group order q is known. This method reduces the given discrete logarithm problem to subgroups of prime power order. This algorithm is very efficient when all the prime factors of the order q are small.

Due to the increase of demand in Cryptology, noteworthy research has been going on. No one knows how long the computation of discrete logarithm problems will remain infeasible. In the meantime, there are significant advances in the DLP based cryptography specially elliptic curve and hyperelliptic curve cryptography which will be continued.

Appendix A: Sage Code for Applications and Attack on Discrete Logarithms

Some basic useful functions

```
def greatest_common_divisor(x, y):
    if(y == 0):
        return abs(x)
    else:
        return greatest_common_divisor(y, x % y)

def my_euler_phi(n):
    result = n
    for p, e in factor(n):
        result *= (1 - 1/p)
    return result

def exponentiation(a,b):
    if b == 0:
        return 1
    if b % 2 == 1:
        return a * exponentiation(a, b//2) ** 2
    else:
        return exponentiation(a, b//2) ** 2

def modular_exponentiation(a,b,n):
    if b == 0:
        return 1
    if b % 2 == 1:
        return (a * modular_exponentiation(a, b//2, n) ** 2) % n
    else:
        return (modular_exponentiation(a, b//2, n) ** 2 ) % n
```

```
print("GCD of two given numbers is:",
      greatest_common_divisor(2^100, 2^110))
```

Output

```
GCD of two given numbers is: 1267650600228229401496703205376
```

```
print("Number of relative primes of the given number:",
      my_euler_phi(2345))
```

Output

```
Number of relative primes of the given number: 1584
```

```
print("12^234 is: ",exponentiation(12,234))
```

Output

```
12^234 is:  337607102894746808086206851969306022575062805800471667328696377526085247029650276133591352965417939210925084674865466980998280377351672751013294193697575298323083242291057917284030246862384631577368553385143225035183580642648151231886430261661009340006 4
```

```
print("1230^4560 (mod 78900) is: ",
      modular_exponentiation(1230,4560,78900))
```

Output

```
1230^4560 (mod 78900) is:  15000
```

Diffie Hellman Key Exchange

```
random_numb = randint(2^130,2^150)
p = next_prime(random_numb)
g= primitive_root(p)
x = randint(1,p-1) # Secret key of Alice
y= randint(1,p-1) # Secret key of Bob

print("Prime          : " , p)
print("primitive root :  " , g)
A = pow(g,x,p)
print("Alice Sends to Bob: " , A)
B = pow(g,y,p)
print("Bob Sends to Alice: ", B)
print()
print("Computation and finding shared secret key:")

key_A = pow(B,x,p)
print("Alice computes: ", key_A)
key_B = pow(A,y,p)
print("Bob   computes: ", key_B)
```

Output

```
Prime          :  5675323474815982609210976487022663692284198 09
primitive root :   3
Alice Sends to Bob:  1167285742992958312829284499846601591386640 64
Bob Sends to Alice:  4029251690900248919857307245165628796297859 39

Computation and finding shared secret key:
Alice computes:  4936328030767512982669483057931362269517631 91
Bob   computes:  4936328030767512982669483057931362269517631 91
```

ElGamal Cryptosystem

```
def Elgamal_key_generation():

    random_numb = randint(2^100, 2^110)
    p = next_prime(random_numb)

    g = primitive_root(p)

    y = randint(1, p-1) # secret_key

    return y, (p, g, pow(g,y,p))

def Elgamal_msg_encryption(m, x, public_key):

    p,g,B = public_key
    C_1 = pow(g,x,p)
    C_2= mod(m*(pow(B,x,p)),p)
    return (C_1, C_2)

def Elgamal_msg_decryption(p, y, cipher_text):
    C_1, C_2 = cipher_text

    C_1_inv = inverse_mod(C_1.lift(),p)
    m = mod((pow(C_1_inv, y, p))*C_2 , p)
    return m
```

```
y, public_key = Elgamal_key_generation()
print("Public key (p, g, B) is :", public_key)
```

Output

```
Public key (p, g, B) is : (112095888253795283848569379735979, 2,
855190133018963517525772306047 70)
```

```
m = 98765432123456789 # hashed message
x = 7654321
cipher_text = Elgamal_msg_encryption(m, x, public_key)
print("Bob receives the pair (C_1, C_2) : ", cipher_text)
```

Output

```
Bob receives the pair (C_1, C_2) :  (365824083172852483798591643 5
6204, 109441759875878125207860712859920)
```

```
print("The expected value of m is:",
      Elgamal_msg_decryption(public_key[0],
                             y, cipher_text))
```

Output

```
The expected value of m is: 98765432123456789
```

El-Gamal signature scheme

```
def Elgamal_signature_key_generation():
    random_numb = randint(2^100, 2^110)
    p = next_prime(random_numb)
    g = primitive_root(p)
    x = randint(1, p-2) # secret_key
    return x, (p, g, pow(g, x, p))

def Elgamal_message_sign(m, x, public_key): #(hash_msg,secret,pub)
    p, g, A = public_key
    k = randint(1, p-2) # secret session_key
    while gcd(k, p-1) != 1:
        k = randint(1, p-1)

    k = mod(k, p-1)
    print("Selected k (secret and only for once):",k)
    t = pow(g,k, p)
    u = (m - x*t.lift())/k
    return (m, t, u)

def Elgamal_signature_verification(public_key, signed_message):
    p, g, A = public_key
    m, t, u = signed_message

    int1 = pow(A,t,p)
    int2 = pow(t,u,p)

    v1 = mod(int1*int2, p)
    v2 = pow(g,m, p)
    if v1 == v2:
        print("He accepts the signature.")
    else:
        print("He rejects the signature")
    return v1, v2, v1==v2
```

```
x, public_key = Elgamal_signature_key_generation()
print("Public key (p, g, A) is :", public_key)
```

Output

```
Public key (p, g, A) is : (1014476830251210901145627101690457, 3,
52887017732245252096353760011592 2)
```

```
m = 1213 # hashed message
signed_message = Elgamal_message_sign(m, x, public_key)
print("Bob receives the triple (m, t, u) : ", signed_message)
```

Output

```
Selected k (secret and only for once): 8923301056937861052938120403547 69
Bob receives the triple (m, t, u) :  (1213, 1725549858685290438435402 8532613, 5878818832005478439421110631301 22)
```

```
Elgamal_signature_verification(public_key, signed_message)
```

Output

```
He accepts the signature.
(55718876817155425531158187112 5549, 55718876817155425531158187112 5549, True)
```

Baby Step Giant Step Method to solve DLP

```
def BSGS(g,a,p):
    m = ceil(sqrt(p))
    baby_steps = [1]
    # computation of Baby steps
    for j in range(0,m):
        baby_steps.append((mod((baby_steps[-1]*g) , p)))
    #print(baby_steps)
    s = baby_steps[-1]
    t = inverse_mod(s.lift(),p)
    A = mod(a,p).lift()
    # computation of Giant steps
    for i in range(0,m):
        r = mod((A*(pow(t,i,p))),p)

        if r in baby_steps:
            print("for i =", i, ", a.g^(-im) (mod p) =" ,  r)
            print("A match has been found for: i =",i, ",",
                  "j =", baby_steps.index(r), "where m =", m)
            x = i*m + baby_steps.index(r)
            print("The solution for the DLP is ", x)
            return  x
    return False
```

```
BSGS(5,283,10103)
```

Output

```
for i = 2 , a.g^(-im) (mod p) = 4599
A match has been found for: i = 2 , j = 32 where m = 101
The solution for the DLP is  234
234
```

In example 3.4, 234 was chosen as secret key which has been revealed now.

```
BSGS(5,57105961,58231351)
```

Output

```
for i = 26 , a.g^(-im) (mod p) = 19566839
A match has been found for: i = 26 , j = 358 where m = 7631
The solution for the DLP is  198764
198764
```

Silver Pohlig Hellman Algorithm to solve DLP

```
# Silver Pohlig Hellman Algorithm to solve DLP
def SPHA(g,a,p):
    prime_factors_list = list(factor(p - 1))
    print("Prime factorization of p-1:", prime_factors_list)
    r = []
    d = []
    for q, n in prime_factors_list:
        print("For x mod " + str(q)+ "^" + str(n) + ":")
        d.append(q^n)
        G = pow(g, (p-1)/(q^n), p)
        B = pow(a, (p-1)/(q^n), p)
        G_inverse = inverse_mod(G.lift(),p)
        x = []
        L = pow(G, q^(n-1), p)
        R = pow(B, q^(n-1), p)

        for x_0 in range(p):
            if pow(L, x_0, p) == mod(R, p):
                x.append(x_0)
                break
        for i in reversed(range(n-1)):
            e = sum([x_i*q^j for x_i, j in zip(x,range(n-i))])
            R = pow((B * (pow(G_inverse,e,p))), q^i, p)
            for xi in range(p):
                if pow(L,xi,p) == mod(R, p):
                    x.append(xi)
                    break
        print("list of x_i's :",x)
        r.append(sum([x_i*q^j for x_i, j in zip(x,range(n))]))

    print("System of congruences:")
    print(r, d)
    solution = crt(r,d)
    print()
    print("The solution of the particular problem is", solution)
    return solution
```

```
SPHA(5,283,10103)
```

Output

```
Prime factorization of p-1: [(2, 1), (5051, 1)]
For x mod 2^1:
list of x_i's : [0]
For x mod 5051^1:
list of x_i's : [234]
System of congruences:
[0, 234] [2, 5051]

The solution of the particular problem is 234
234
```

```
SPHA(3,2,65537)
```

Output

```
Prime factorization of p-1: [(2, 16)]
For x mod 2^16:
list of x_i's : [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1]
System of congruences:
[55296] [65536]

The solution of the particular problem is 55296
55296
```

```
# To verify the obtained solution:
pow(3,55296,65537) == 2
```

Output

```
True
```

Diffie Hellman Key Exchange for Elliptic Curve

```
p = next_prime(2^20) # Prime
E = EllipticCurve(GF(p), [15,101])
G= E([315424,602724])
x = randint(1,p-1) # Secret key of Alice
y= randint(1,p-1) # Secret key of Bob
print(E)
print("Base point: " , G)
A = x*G
print("Alice Sends to Bob: " , A)
B = y*G
print("Bob Sends to Alice: ", B)
print("Computation and finding shared secret key:")
key_A = x*B
print("Alice computes shared secret point: ", key_A)
key_B = y*A
print("Bob   computes shared secret point: ", key_B)
```

Output

```
Elliptic Curve defined by y^2 = x^3 + 15*x + 101 over Finite
Field of size 1048583
Base point:  (315424 : 602724 : 1)
Alice Sends to Bob:  (566576 : 690141 : 1)
Bob Sends to Alice:  (624973 : 386887 : 1)

Computation and finding shared secret key:
Alice computes shared secret point:  (627989 : 273894 : 1)
Bob   computes shared secret point:  (627989 : 273894 : 1)
```

Elliptic Curve ElGamal Cryptosystem

```
def EC_Elgamal_key_generation():
    p = next_prime(2^31) # Prime
    a = 12; b = 456
    E= EllipticCurve(GF(p),[a,b])
    G = E([358356489,1550454291]) # base_point
    y = 106 # randint(1, p-1) # secret_key
    B = y*G
    return y, (p,a,b,G,B)

def EC_Elgamal_msg_encryption(m, x, public_key):
    p,a,b,G,B = public_key
    E = EllipticCurve(GF(p),[a,b])
    m = E(m)
    G = E(G)
    C_1 = x*G
    C_2= m + x*B
    return (C_1, C_2)

def EC_Elgamal_msg_decryption(y, cipher_text):
    C_1, C_2 = cipher_text
    m = C_2 - y*C_1
    return m
```

```
y, public_key = EC_Elgamal_key_generation()
print("Publicly available information: E,(p,a,b, G, B) is :",
      public_key)
```

Output

```
Publicly available information: E,(p,a,b, G, B) is :
(2147483659, 12, 456, (358356489 : 1550454291 : 1),
(1251387476 : 107845780 : 1))
```

```
m = (317518005,1403368156)
x = 147
cipher_text = EC_Elgamal_msg_encryption(m, x, public_key)
print("Bob receives the pair (C_1, C_2) : ", cipher_text)
```

Output

```
Bob receives the pair (C_1, C_2) :
((1698475745 : 1824909440 : 1), (991335935 : 523312982 : 1))
```

```
print("The expected value of m is:",
      EC_Elgamal_msg_decryption(y, cipher_text))
```

Output

```
The expected value of m is: (317518005 : 1403368156 : 1)
```

```
p = next_prime(2^70)
E= EllipticCurve(GF(p),[15,101])
print(E)
print("factors of order of E: ", factor(E.order()))
P = E.random_point()
print("P:", P)
print("factors of order of point P: ",factor(P.order()))
```

Output

```
Elliptic Curve defined by y^2 = x^3 + 15*x + 101 over Finite
Field of size 1180591620717411303449
factors of order of E:  3 * 11 * 29 * 1233638057147363047
P: (257131113214664382390 : 571545366858964121582 : 1)
factors of order of point P:  3 * 11 * 29 * 1233638057147363047
```

```
G = 957*P
print(G)
q = G.order()
print("order of point G:", q)
print(GF(q))
```

Output

```
(238365256287691324451 : 982706285824234617255 : 1)
order of point G: 1233638057147363047
Finite Field of size 1233638057147363047
```

Elliptic Curve Digital Signature Algorithm.

```
def ECDSA_key_generation():
    p = next_prime(2^70)
    a = 15
    b= 101
    E= EllipticCurve(GF(p),[a,b])
    G = E([23836525628769132445 1,98270628582423461725 5])
    q = 1233638057147363047 # prime order
    x = 110 # randint(1, p-2) # secret_key
    A = x*G
    return x, (p,a,b, G,q,A)

def ECDSA_signed_message(m, x, public_info): # (msg,secret,pub.)
    q=3577
    p,a,b, G,q,A = public_info
    k =  randint(1, q-1) # session_key
    #while gcd(k, q) != 1:
    #    k = randint(1, q-1)
    print("Selected secret k:",k)

    T = k*G
    x_t = T[0]
    k_inv = inverse_mod(k, q)
    u = mod(k_inv*(m + x* x_t.lift()),q)
    return (m, T, u)

def ECDSA_signature_verification(public_info, signed_message):
    p,a,b, G,q,A = public_info
    m, T, u = signed_message
    x_t = T[0].lift()
    u_inv = inverse_mod(u.lift(),q)
    v1 = (mod(m*u_inv, q)).lift()
    v2 = (mod(x_t*u_inv,q)).lift()
    V= v1*G +v2*A

    if V == T:
        print("He accepts the signature.")
    else:
        print("He rejects the signature")
    return V
```

```
x, public_info = ECDSA_key_generation()
print("Public key (p,a,b, G,q,A) is :", public_info)
```

Output

```
Public key (p,a,b, G,q,A) is : (1180591620717411303449, 15, 101,
(238365256287691324451 : 982706285824234617255 : 1), 12336380571
47363047, (659741503912424842005 : 615073760674055601946 : 1))
```

```
m = 2345678901 # hashed message
signed_message = ECDSA_signed_message(m, x, public_info)
print("Bob receives the triple (m, T, u) : ", signed_message)
```

Output

```
Selected secret k: 447525510508179451
Bob receives the triple (m, T, u) :  (2345678901, (92530604647012
0288933 : 59554930340084291803 : 1), 601113646893181433)
```

```
ECDSA_signature_verification(public_info, signed_message)
```

Output

```
He accepts the signature.
(925306046470120288933 : 59554930340084291803 : 1)
```

Baby Step Giant Step Method to solve ECDLP

```
def BSGS_ECDLP(P, Q, E):
    if P == Q:
        return 1
    m = ceil(sqrt(P.order()))
    baby_list = [j*P for j in range(m)]
    #print(baby_list)

    for i in range(m):
        result = Q - (i*m)*P
        #print("Q - (i*m)*P:", result)
        if result in baby_list:
            print("A match has been found for: i =",i, ",",
                  "j =", baby_list.index(result), "where m =", m)
            x = (i*m + baby_list.index(result)) % P.order()
            print("The solution for ECDLP is ", x)
            return  x
    return False
```

```
E = EllipticCurve(GF(5683), [5, 89])
P = E([135,20]); Q = E([1746,2547])
BSGS_ECDLP(P, Q, E)
```

Output

```
A match has been found for: i = 1 , j = 30 where m = 76
The solution for ECDLP is  106
106
```

```
# To verify the solution
106*P == Q
```

Output

```
True
```

In example 4.9, 106 was chosen as secret key which has been revealed now.

Silver Pohlig Hellman Method to solve ECDLP

```
def SPHA_ECDLP(E, P, Q):
    q = P.order()
    prime_factors_list = list(factor(q))
    print("List of prime factorization:", prime_factors_list)
    r = []
    for p, a in prime_factors_list:
        print("For x mod " + str(p)+ "^" + str(a) + ":")
        W = E(0, 1, 0)
        x_0 = 0
        P_m = (q // p) * P
        v = 0
        x=[]
        for k in range(0, a):
            W += int(x_0 * (p^(k - 1))) * P
            Q_m = int(q // (p^(k + 1))) * (Q - W)

            for x_0 in range(q):

                if x_0 * P_m  == Q_m:
                    x.append(x_0)
                    break

            v += x_0 * pow(p, k)
        print("List of x_i's :", x)
        r.append(int(mod(v, p^a)))

    d = [p^a for p,a in prime_factors_list]
    print("System of congruences:")
    print(r, d)
    solution = crt(r,  d)

    print("The solution of the particular problem is", solution)
    return solution
```

```
p = 599 ; E = EllipticCurve(GF(p), [0, 1]) ;
P = E([60,19]) ; Q = E([277,239])

SPHA_ECDLP(E, P, Q)
```

Output

```
List of prime factorization: [(2, 3), (3, 1), (5, 2)]
For x mod 2^3:
List of x_i's : [0, 1, 0]
For x mod 3^1:
List of x_i's : [2]
For x mod 5^2:
List of x_i's : [1, 3]

System of congruences:
[2, 2, 16] [8, 3, 25]
The solution of the particular problem is 266
266
```

```
# To verify the solution
266* E([60,19]) == E([277,239])
```

Output

```
True
```

Bibliography

[1] Leonard Adleman. A subexponential algorithm for the discrete logarithm problem with applications to cryptography. In *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*, pages 55–60, 1979.

[2] Eric Bach, Jeffrey Outlaw Shallit, Jeffrey Shallit, and Shallit Jeffrey. *Algorithmic number theory: Efficient algorithms*, volume 1. MIT press, 1996.

[3] Ian Blake, Gadiel Seroussi, and Nigel Smart. *Elliptic Curves in Cryptography*. Cambridge University Press, 1999.

[4] Joppe W. Bos, Marcelo E. Kaihara, Thorsten Kleinjung, Arjen K. Lenstra, and Peter L. Montgomery. On the security of 1024-bit rsa and 160-bit elliptic curve cryptography. *IACR Cryptol. ePrint Arch.*, 2009:389, 2009.

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

[6] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

[7] Steven D. Galbraith. *Mathematics of Public Key Cryptography*. Cambridge University Press, 2012.

[8] Darrel Hankerson, Scott A. Vanstone, and Alfred Menezes. Guide to elliptic curve cryptography. In *Springer Professional Computing*, 2004.

[9] Dale Husemoeller. *Elliptic Curves*. Graduate Texts in Mathematics. Springer, Second Edition.

[10] Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ecdsa). *Int. J. Inf. Secur.*, 1(1):36â63, aug 2001.

[11] J. Katz and Y. Lindell. *Introduction to Modern Cryptography*. Chapman & Hall/CRC Cryptography and Network Security Series. CRC Press, 2014.

[12] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of computation*, 48(177):203–209, 1987.

[13] Neal Koblitz. *Inroduction to Elliptic Curves and Modular Forms*. Graduate Texts in Mathematics. Springer-Verlag, Second Edition.

[14] Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone. *Handbook of applied cryptography*. CRC press, 2018.

[15] Ralph C. Merkle. Secure communications over insecure channels. *Commun. ACM*, 21(4):294â299, apr 1978.

[16] Victor S. Miller. Use of elliptic curves in cryptography. In Hugh C. Williams, editor, *Advances in Cryptology — CRYPTO '85 Proceedings*, pages 417–426, Berlin, Heidelberg, 1986. Springer Berlin Heidelberg.

[17] Christof Paar and Jan Pelzl. *Understanding Cryptography: A Textbook for Students and Practitioners*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[18] S. Pohlig and M. Hellman. An improved algorithm for computing logarithms over gf(p) and its cryptographic significance (corresp.). *IEEE Transactions on Information Theory*, 24(1):106–110, 1978.

[19] J. Pollard. Kangaroos, monopoly and discrete logarithms. *Journal of Cryptology*, 13:437–447, 01 2000.

[20] John M. Pollard. Monte carlo methods for index computation (). *Mathematics of Computation*, 32:918–924, 1978.

[21] Daniel Shanks. Class number, a theory of factorization, and genera. 1971.

[22] M. Stamp and R. M. Low. *Applied cryptanalysis: Breaking ciphers in the real world*. Hoboken, N.J.: Wiley-Interscience, 2007.

[23] Douglas R Stinson. *Cryptography: theory and practice*. Chapman and Hall/CRC, 2005.

[24] W. Trappe and L.C. Washington. *Introduction to Cryptography: With Coding Theory*. Prentice Hall, 2002.

[25] Song Yuan Yan. Quantum computational number theory. In *Springer International Publishing*, 2015.

Erklärung

Hiermit erkläre ich, dass ich meine Arbeit selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die Arbeit noch nicht anderweitig für Prüfungszwecke vorgelegt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Mittweida, im February 2022