Elliptic Curve Cryptography

Anita Dash, Kethari Narasimha Vardhan, Tata Sai Manoj

A Report Submitted to

Indian Institute of Technology Hyderabad

As Part of a Credited Research Project

Under Guidance of Dr. Pradipto Banerjee

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Department of Mathematics

May 13, 2023

All of the implementations can be found at [1] [2] and [3]

Chapter 1

Introduction

1.1 Weierstrass Equation

The Weierstrass form of an elliptic curve E over a field K is given by the equation

$$y^2 = x^3 + Ax + B \tag{1.1}$$

with $A, B \in K$ and $4A^3 + 27B^2 \neq 0$.

1.2 Group Law

Let E be an elliptic curve defined by $y^2 = x^3 + Ax + B$, and let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be points on E with $P_1, P_2 \neq \infty$. Define $P_1 + P_2 = P_3 = (x_3, y_3)$ as follows:

1. If $x_1 \neq x_2$, then

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

   where $m = (y_2 - y_1)/(x_2 - x_1)$.

2. If $x_1 = x_2$ but $y_1 \neq y_2$, then $P_1 + P_2 = \infty$.

3. If $P_1 = P_2$ and $y_1 \neq 0$, then

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

where $m = (3x_1^2 + A)/(2y_1)$

4. If $P_1 = P_2$ and $y_1 = 0$, then $P_1 + P_2 = \infty$.

We know that $P_1, P_2 \in E$ and $(P_1 + P_2) \in E$. Therefore $E(K)$ is closed under the addition of above points.

Definition 1. *We define the point at infinity as the additive identity element of the group law on the curve.*

$$P + \infty = P \quad \forall P \in E \tag{1.2}$$

Theorem 1.2.1. *The addition of points on an elliptic curve E satisfies the following properties:*

1. *(Commutativity) $P_1 + P_2 = P_2 + P_1$ for all P_1, P_2 on E.*

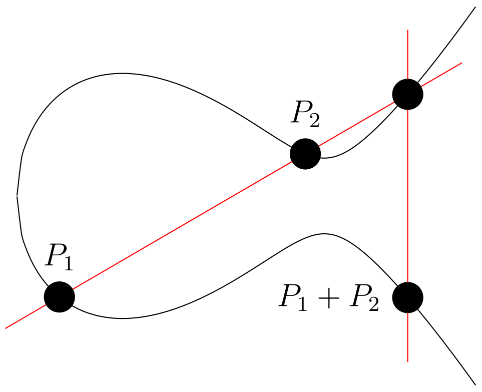
2. *(Existence of identity) $P + \infty = P$ for all points P on E.*

3. *(Existence of inverses) Given P on E, there exists P' on E with $P + P' = \infty$. This point P' will usually be denoted $-P$.*

4. *(Associativity) $(P_1 + P_2) + P_3 = P_1 + (P_2 + P_3)$ for all P_1, P_2, P_3 on E.*

The points on E form an additive Abelian group with ∞ as the identity element.

Proof. The line through P1 to P2, is the same as the line through P2 to P1. Therefore the point $(P_1 + P_2)$ will be the same as the point $(P_2 + P_1)$. Thus it is commutative. The identity property of ∞ holds by definition. Let P' be the reflection of the point P with respect to x axis. Then $P + P' = \infty$. Thus inverse exists for a given point P in E. The proof of associativity can be found here [1]. ☐



1.2.1 Integer times a Point

If we need to calculate the point kP, performing k additions of P can be computationally costly. To solve this problem, we can use the algorithm of successive doubling to compute kP for a point P on the Elliptic curve E defined over the field K.

Algorithm 1 Successive doubling to find kP

```
1:  procedure MULTIPLY($k, P$)                    ▷ to find the point $kP$
2:      $a \leftarrow k$
3:      $B \leftarrow \infty$
4:      $C \leftarrow P$
5:      if $a \bmod 2 = 0$ then
6:          $a \leftarrow a/2$
7:          $B \leftarrow B$
8:          $C \leftarrow C + C$
9:      else
10:         $a \leftarrow a - 1$
11:         $B \leftarrow B + C$
12:         $C \leftarrow C$
13:     end if
14:     if $a \neq 0$ then
15:         Go to Step 5
16:     end if
17:     return $B$                                ▷ $kP = B$
18: end procedure
```

1.3 Endomorphism

Definition 2. *let α be a homomorphism. $\alpha : E(\bar{K}) \to E(\bar{K})$ that is given by rational functions $R_1(x, y), R_2(x, y)$ with coefficients in $\bar{K}$, such that*

$$\alpha(x, y) = (R_1(x, y), R_2(x, y)) \quad \forall (x, y) \in E(\bar{K})$$

Then α is an endomorphism of E.

1.3.1 Degree of α

Below is a standard form for the rational functions describing an endomorphism α of an elliptic curve given in Weierstrass form.

$$\alpha(x, y) = (r_1(x), r_2(x)y)$$

where $r_1(x)$ and $r_2(x)$ are rational functions and

$$r_1(x) = \frac{p(x)}{q(x)}$$

where the polynomials $P(x)$ and $q(x)$ do not have a common factor.

Definition 3. *We define degree of α to be*

$$\deg(\alpha) = Max\{\deg(p(x)), \deg(q(x))\}$$

When α is non-trivial. If $\alpha = 0$, then $\deg(\alpha) = 0$.

1.3.2 Separable Endomorphism

Definition 4. *$\alpha \neq 0$ is said to be a separable endomorphism if the derivative $r_1'(x)$ is not identically zero*

1.3.3 Frobenius Map

Definition 5. *Suppose E is an elliptic curve defined over the finite field $\mathbb{F}_q$. Let*

$$\phi_q(x, y) = (x^q, y^q) \tag{1.3}$$

Then ϕ_q is the Frobenius map

Lemma 1.3.1. *Let E be defined over $\mathbb{F}_q$. Then ϕ_q is an endomorphism of E of degree q, and ϕ_q is not separable.*

Proof. Let $(x_1, y_1), (x_2, y_2) \in E(\mathbb{F}_q)$ with $x_1 \neq x_2$
Let $(x_3, y_3) \in E(\mathbb{F}_q)$ such that $(x_3, y_3) = (x_1, y_1) + (x_2 + y_2)$

$$\phi_q(x_3, y_3) = (x_3^q, y_3^q)$$

Using the definition in 1.2
$$m^q = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^q$$
all the coefficients in the expansion except for y_2^q, y_1^q, x_2^q, x_1^q will be a multiple of q, therefore
$$m^q = \frac{y_2^q - y_1^q}{x_2^q - x_1^q}$$
let $m' = m^q$, then
$$x_3^q = (m^2 - x_1 - x_2)^q$$
all the coefficients in the expansion except for m^{2q}, x_1^q, x_2^q will be a multiple of q, therefore
$$x_3^q = m'^2 - x_1^q - x_2^q$$
similarly we get,
$$y_3^q = m'(x_1^q - x_3^q) - y_1^q$$
Thus,
$$\phi_q(x_3, y_3) = \phi_q(x_1, y_1) + \phi_q(x_2, y_2)$$
We can check for the other cases similarly.
Therefore ϕ_q is a homomorphism given by rational functions and thus is an endomorphism of E.
qx^{q-1} in $\mathbb{F}_q$ is zero as q is zero in $\mathbb{F}_q$. Therefore derivative of x^q is identically zero. Hence ϕ_q is not separable. □

Proposition 1.3.1. *Let $\alpha \neq 0$ be a separable endomorphism of an elliptic curve E. Then*
$$\deg(\alpha) = \#\operatorname{Ker}(\alpha),$$
where $\operatorname{Ker}(\alpha)$ is the kernel of the homomorphism $\alpha : E(K) \to E(K)$. If $\alpha \neq 0$ is not separable, then
$$\deg(\alpha) > \#\operatorname{Ker}(\alpha)$$

Proof. Refer to (1) for the proof. □

1.4 Torsion Points

Let E be an elliptic curve defined over field K. Let n be a positive integer then,

$$E[n] = \{P \in E(K) \mid nP = \infty\}$$

Theorem 1.4.1. *Let E be an elliptic curve over a field K and let n be a positive integer. If the characteristic of K does not divide n, or is 0, then*

$$E[n] \cong \mathbb{Z}_n \oplus \mathbb{Z}_n$$

If the characteristic of K is $p > 0$ and $p \mid n$, write $n = p^r n'$ with $p \nmid n'$. Then

$$E[n] \cong \mathbb{Z}_{n'} \oplus \mathbb{Z}_n \quad \textit{or} \quad \mathbb{Z}_n \oplus \mathbb{Z}_{n'}$$

1.5 Elliptic Curves over Finite Fields

Let $\mathbb{F}$ be a finite field. Let E be an elliptic curve defined over $\mathbb{F}$. Since there are only finitely many possible pairs $(x, y) \quad s.t \quad x, y \in \mathbb{F}$, The group $E(\mathbb{F})$ is finite.

Theorem 1.5.1. *Let E be an elliptic curve over a field $\mathbb{F}_q$. Then*

$$E(\mathbb{F}_q) \cong \mathbb{Z}_n \quad \textit{or} \quad \mathbb{Z}_{n_1} \oplus \mathbb{Z}_{n_2}$$

For some integer $n \geq 1$, or for some integers $n_1, n_2 \geq 1$ with n_1 dividing n_2.

Theorem 1.5.2. *[Hasse's Theorem] Let E be an elliptic curve over a field $\mathbb{F}_q$. Then the order of $E(\mathbb{F}_q)$ satisfies*

$$|q + 1 - \#E(\mathbb{F}_q)| \leq 2\sqrt{q}$$

Proof. Refer to (1) for the above proofs. □

1.5.1 Order of Points

Let P be a point on the Elliptic curve E over a field $\mathbb{F}_q$.
Let $\#E(\mathbb{F}_q) = N$, By Lagrange's theorem, we know that $NP = \infty$.
By using Hasse's theorem we can try all values in the range $(q+1-2\sqrt{q}, q+1+2\sqrt{q})$ and check which one satisfies. This takes around $4\sqrt{q}$ steps.
We can speed up to around $4q^{1/4}$ steps using the baby step, giant step method to find the order of point P.

Algorithm 2 Baby Step, Giant Step method to find order of point P

```
 1: procedure BSGSORDER($P$)                          ▷ Compute order of point P
 2:     $Q \leftarrow (q+1)P$
 3:     Choose an integer $m$ such that, $m \geq q^{\frac{1}{4}}$
 4:     for $j = 0$ to $m$ do
 5:         compute and store $jP$
 6:     end for
 7:     for $k = -m$ to $m$ do
 8:         compute $Q + k(2mP)$
 9:         if $Q + k(2mP) = \pm jp$ then
10:             break                                ▷ $(q+1+2mk \mp j)P = \infty$
11:         end if
12:     end for
13:     $M \leftarrow (q+1+2mk) \mp j$
14:     Factor $M$, $p_1, ...p_r$ are the distinct prime factors of $M$
15:     for $i = 1$ to $r$ do
16:         $val \leftarrow (\frac{M}{p_i})P$
17:         if $val = \infty$ then
18:             $M \leftarrow (\frac{M}{p_i})$
19:             go to step 14
20:         else
21:             break
22:         end if
23:     end for
24:     return $M$                                    ▷ order of $P$ is $M$
25: end procedure
```

Correctness of Baby Step, Giant Step Method to find order of P

Lemma 1.5.1. *Let a be an integer with $|a| \leq 2m^2$. There exists integers a_0 and a_1 with $-m \leq a_0, a_1 \leq m$ such that*

$$a = a_0 + 2ma_1$$

Proof. let $a_0 \equiv a \pmod{2m}$, with $-m \leq a_0 \leq m$. let $a_1 = \frac{(a-a_0)}{2m}$

$$|a - a_0| \leq |a| + |a_0| \leq 2m^2 + m$$

$$\implies |a_1| \leq \frac{2m^2+m}{2m} < m+1$$

$$-m \leq a_1 \leq m$$

□

According to Hasse's theorem:

$$q + 1 - 2\sqrt{q} \leq N \leq q + 1 + 2\sqrt{q}$$

$$\implies N = q + 1 - a \quad \textit{with} \quad |a| \leq 2\sqrt{q}$$

At step-7 of the algorithm let $k = -a_1$ (k lies in between $-m$ to m), Then using 1.5.1, there exists an integer $a_0 \in [-m, m]$

$$Q - a_1(2mP) = (q + 1 - 2ma_1)P = (q + 1 - a + a_0)P = NP + a_0P = a_0P$$

Therefore at step-8 of the algorithm for $j = |a_0|$ there is a match.

Lemma 1.5.2. *Let G be an additive group (with identity element 0) and let $g \in G$. Suppose $Mg = 0$ for some positive integer M. Let $p_1, ..., p_r$ be the distinct primes dividing M. if $(M/p_i)g \neq 0$ for all i, then M is the order of g.*

Proof. let k be the order of g. The $k|M$ as $Mg = 0$. Let $k \neq M$.
Let p_i be a prime divisor of M/k.

$$\implies p_ik|M \implies k|(M/P_i) \implies (M/P_i)g = 0$$

Which is a contradiction, as we assumed $(M/p_i)g \neq 0$ for all i.
Therefore $k = M$ □

Therefore Steps 15 to 21 find the order of P.

Chapter 2

The Discrete Logarithm Problem

2.1 Classical Discrete Logarithm Problem

Let p be a prime. Let $a, b \in \mathbb{Z}$, $a \pmod{p} \neq 0$, $b \pmod{p} \neq 0$. Suppose there exists $k \in \mathbb{Z}$ such that,

$$a^k \equiv b \pmod{p}$$

The Classical Discrete problem is to find the k that satisfies the above condition.

2.2 Discrete Logs on Elliptic Curves

Let us consider the group $E(\mathbb{F}_q)$ for some elliptic curve E over finite field $\mathbb{F}_q$. Let $a, b \in E$. Then the discrete logarithm problem is to find an integer k such that

$$ka = b$$

2.3 General Attacks on Discrete Logs

We can attack a discrete log problem by using a simple brute force method - trying all possible values of k until one satisfies the condition. However this method is very impractical and expensive as k can be of very huge size (several hundred digits). Below are a few methods that can be used to solve the discrete log problem in a more practical way.

We consider the group: an Elliptic curve E over the field $\mathbb{F}_q$ with order N. We are given the points $P, Q \in E(\mathbb{F}_q)$ and we trying to solve:

$$kP = Q$$

We assume that such k exists.

2.3.1 Baby Step, Giant Step

This method was developed by D. Shanks.
It requires approximately $\sqrt{N}$ steps and around $\sqrt{N}$ storage. (1)

Algorithm 3 Baby Step, Giant Step method to find k

```
1: procedure BSGSDLP($P, Q$)                          ▷ Find $k$ such that, $kP = Q$
2:     choose integer $m$ such that $m \geq \sqrt{N}$
3:     compute mP
4:     for $i = 0$ to $m - 1$ do
5:         compute and store $iP$
6:     end for
7:     for $j = 0$ to $m$-1 do
8:         compute $Q - jmP$
9:         if $Q - jmP = ip$ then
10:            $k \leftarrow (i + jm) \pmod{N}$
11:            break
12:        end if
13:    end for
14:    return $k$
15: end procedure
```

Correctness of Baby Step, Giant Step Method

We know that $m^2 \geq N$
Let $0 \leq k \leq m^2$ and let k_0 be an integer such that $0 \leq k_0 < m$ and $k \equiv k_0 \pmod{m}$
let $k_1 = (k - k_0)/m \implies k_1 < (m^2 - 0)/m \implies 0 \leq k_1 < m$
In step-8 of the above algorithm let $j = k_1$

$$Q - k_1 mP = kP - k_1 mP = (k - k + k_0)P = k_0 P$$

Therefore when $j = k_1$ and $i = k_0$, There is a match
Note: We were able to substitute Q with kP because we have assumed that such k exists.

2.3.2 Pollard's ρ Method

We first choose a positive integer s and divide the elliptic curve into s disjoint subsets $S_1, S_2, ..., S_s$ of approximately same size.
We then choose $2s$ random integers $a_i, b_i \pmod N$. Let

$$M_i = a_iP + b_iQ \quad \forall i \in \{1, 2, ..., s\}$$

. We define $f : E(\mathbb{F}_q) \to E(\mathbb{F}_q)$ such that,

$$f(g) = g + M_i \quad if \quad g \in S_i$$

We start with a random point P_0 and compute the iterations $P_{i+1} = f(P_i)$. We also need to keep track of how the points are expressed in terms of P and Q. Since $E(\mathbb{F}_q)$ is a finite set, there will be some indices $i_0 < j_0$ such that $P_{i_0} = P_{j_0}$. Let,

$$P_{i_0} = u_{i_0}P + v_{i_0}Q$$
$$P_{j_0} = u_{j_0}P + v_{j_0}Q$$

Therefore,

$$u_{j_0}P + v_{j_0}Q = u_{i_0}P + v_{i_0}Q$$
$$(u_{i_0} - u_{j_0})P = (v_{j_0} - v_{i_0})Q$$

let $\gcd((v_{j_0} - v_{i_0}), N) = d$, then we have

$$k \equiv (v_{j_0} - v_{i_0})^{-1}(u_{i_0} - u_{j_0}) \pmod{N/d}$$

This gives us d choices for k, Usually d will be small, so we try all possibilities until we have $Q = kP$.

Algorithm 4 Pollard's ρ Method to find k

```
procedure PRHO($P, Q$)                          ▷ Find $k$ such that, $kP = Q$
    choose integer $s$
    for $i = 0$ to $s - 1$ do
        choose random integers $a_i$, $b_i \pmod N$
        $M_i \leftarrow a_i P + b_i Q$
    end for
    choose random integers $a_0, b_0 \pmod N$
    $P_0 \leftarrow a_0 P + b_0 Q$
    for $n = 0$ to $N - 1$ do
        $f(P_n) \leftarrow P_n + M_j$      ▷ if $x$ is the x-coordinate of $P_n$, then $x \equiv j \pmod s$
        $P_{n+1} \leftarrow f(P_n)$         ▷ also store $u_{n+1}, v_{n+1}$ s.t $P_{n+1} = u_{n+1}P + v_{n+1}Q$
        if $P_{n+1} = P_j$ then                 ▷ For some $j$ in 1 to $n$
            $k \leftarrow (v_j - v_{n+1})^{-1}(u_{n+1} - u_j) \pmod N$
            break
        end if
    end for
    return $k$
end procedure
```

The above implementation takes around $\sqrt{N}$ storage which is similar to Baby Step, Giant step Method. R. W. Floyd found an algorithm that does much better at the cost of a little more computation.
The key idea is the compute pairs (P_i, P_{2i}) for $i = 1, 2, ...$, but only store the current pair. Calculated by the rules:

$$P_{i+1} = f(P_i), \quad P_{2(i+1)} = f(f(P_{2i}))$$

We know that once if there's a match for two indices differing by d, all subsequent indices differing by d will yield matches.
Let $P_{i_0} = P_{j_0}$ and $(j_0 - i_0) = d$, let i be a multiple of d and $i_0 \leq i \leq j_0$. then the indices $i, 2i$ differ by d and hence yield a match.

Time Complexity

For a random function f, the tail and cycle length of the "rho" is $\sqrt{\frac{\pi N}{8}}$, and the proof can be found here [2]. So, the time complexity will be $O(\sqrt{N})$.

2.3.3 Pohlig-Hellman Method

Let N be the order of P , and the prime factorization of N,

$$N = \prod_i q_i^{e_i}$$

We find $k \pmod{q_i^{e_i}}$ for each i using the algorithm below and then use Chinese Remainder theorem to combine these and obtain $k \pmod N$.

Algorithm 5 Pohlig-Hellman Method to find $k \pmod{q^e}$

```
 1: procedure POHLIG-HELLMAN($P, Q$)
 2:     Compute $T = \{j(\frac{N}{q}P) \mid 0 \le j \le q-1\}$
 3:     Compute $\frac{N}{q}Q$
 4:     find $k_0$ such that $\frac{N}{q^r}Q = k_0(\frac{N}{q}P)$
 5:     for $r = 1$ to $e-1$ do
 6:         $Q_r = Q_{r-1} - k_{r-1}q^{r-1}P$
 7:         Find $k_r$ such that $\frac{N}{q^{r+1}}Q_r = k_r(\frac{N}{q}P)$
 8:     end for
 9:     $k \leftarrow (k_0 + k_1 q + ... + K_{e-1}q^{e-1}) \pmod{q^e}$
10:     return $k$
11: end procedure
```

Correctness of Pohlig-Hellman Method

The base q expansion of k is $k_0 + k_1 q + k_2 q^2 + ...$
In step-3:

$$\begin{aligned}\frac{N}{q}Q &= \frac{N}{q}(k_0 + q(k_1 + k_2 q + ...))P \\ &= k_0\frac{N}{q}P + (k_1 + k_2 q + ...)NP = k_0\frac{N}{q}P\end{aligned}$$

Therefore in Step-4 we find k_0
Similarly in Steps 5 to 8 we find $k_1, k_2, ...k_{e-1}$ (We do not go beyond $e-1$ as we are finding the value $k \pmod{q^e}$ and thus all those values will be 0)
In Step-6 we compute $k \pmod{q^e}$ using its expansion in base q

Time Complexity

The research paper [(3)] suggests that the time complexity is $O(\sqrt{N})$.

Chapter 3

Cryptography

3.1 Introduction

Cryptography, as we see it, is the study of having secure communication from outsider observers. It includes three main aspects in it, key generation, encryption algorithm, and decryption algorithm, and the combination is called a cryptosystem. The main objective includes Confidentiality, Non-repudiation, Integrity, and Authenticity. The study of cryptography began with ciphers which are still the building blocks for modern-day cryptosystems. The current cryptosystems and their algorithms are much more complex and use multiple rounds of ciphers to encrypt the message securely. We have two major cryptosystems, namely symmetric (private key) and asymmetric (public key). In this paper, we are going to discuss and compare various algorithms in asymmetric cryptosystems, while majorly focusing on the Elliptic Curve Cryptosystem. A few examples of asymmetric cryptosystems include

- RSA (Rivest-Shamir-Adleman)
- ECC (Elliptic Curve Cryptography)
- Diffie-Hellman Key Exchange
- ElGamal Key Exchange
- Digital Signature Algorithm
- Key Serialization

RSA Key size (in bits)	ECC Key size (in bits)
1024	160
2048	224
3072	256
7680	384
15360	521

Table 3.1: Key Comparison between RSA and ECC

3.2 History of Asymmetric Cryptosystems

One of the most classical examples of asymmetric cryptography first published in 1977 by three scientists Ron Rivest, Adi Shamir and Leonard Adleman of the Massachusetts Institute of Technology, is the RSA. Since then, this cryptosystem has bloomed exponentially. Lately, as the power of computers has increased and become vulnerable to modern technology attacks, cryptographers have started to look out for stronger encryption-decryption schemes. Here's when the Elliptic Curve Cryptography came into the picture. While the RSA system is mainly built over the integer factorization problem, the ECC system is built on the difficulty in solving the Discrete Logarithm Problem on Elliptic Curves. Various cryptoanalysts are predicting that ECC will be overtaking RSA completely as the scalability of RSA is looming. The major reason for this shift is acquiring the same level of security with less memory, less time, and more efficiency. The researchers have claimed that the security that 1024-bit RSA provides, can be provided by ECC in just 164-bit. Another great addition to the ECC is that it is highly scalable and can be used in conjunction with other asymmetric schemes such as DSA, Diffie-Hellman, ElGamal, etc. ECC is **FIPS-certified** and is also endorsed by the National Security Agency (NSA).

A small glimpse of how ECC is better in comparison to the classical RSA can be seen in Table 3.1 above. An RSA of 1024-bit length used to be secure but is no longer considered secure. There are recent claims from Chinese researchers that the 2048-bit RSA could be cracked using their algorithm. Therefore, it is high time to shift to ECC from RSA. It is also to be noted that the shorter key lengths mean devices require less processing power to encrypt and decrypt data, making ECC a good fit for mobile devices, the Internet of Things, and other use cases with more limited computing power.

3.3 Encryption-Decryption Schemes

ECC is replacing RSA and various other asymmetric cryptographic schemes due to its various advantages and scalability. Here, we try to look at how ECC is used in conjunction with Massey-Omura, Diffie-Hellman, ElGamal, and DSA.
We now discuss various encryption-decryption schemes involving Elliptic Curves.

3.3.1 Diffie-Hellman Key Exchange

It is a mathematical method of exchanging cryptographic keys and is one of the earliest practical examples of public key exchanges implemented in the field of cryptography. The main difference between DH (Diffie-Hellman) and ECDH (Elliptic Curve Diffie-Hellman) is the group that is being chosen to compute the secret key(s). While the regular Diffie-Hellman uses a multiplicative group of integers modulo a prime p, the Elliptic Curve Diffie-Hellman uses a multiplicative Abelian group of points of an Elliptic Curve over field F_q.

Let us take Alice and Bob who wish to communicate;
(Note: G is the generator point of the curve with order n)

Algorithm 6 Elliptic Curve Diffie-Hellman

```
1: procedure ECDH($C, G$)                              ▷ $C : y^2 = x^3 + ax + b\ (F_q)$
2:     $A$ chooses a private key $1 \leq k_A \leq n - 1$
3:     $B$ chooses a private key $1 \leq k_B \leq n - 1$
4:     Compute $P_A = k_A G$ and share with $B$
5:     Compute $P_B = k_B G$ and share with $A$
6:     $A$ computes $P_{AB} = k_A(k_B G)$
7:     $B$ computes $P_{BA} = k_B(k_A G)$
8:     return $P_{AB}$ or $P_{BA}$                        ▷ Shared private key
9: end procedure
```

We try to check the time complexity of the above algorithm.
Steps 2, 3 take $O(1)$ time as we generate integers $1 \leq k_A, k_B \leq n - 1$.
Steps 4, 5, 6, 7 take $O(N^{k+1})$ time as the point multiplication depends on the multiplication algorithm used, where N is the bit length of prime n. Therefore, the time complexity for ECDH is

$$\text{T.C} = O((log\, n)^{k+1})$$

However, please note that the ECDH time complexity might vary depending on the specific algorithms used and how they are implemented in the intermediate steps.

3.3.2 Massey Omura Algorithm

This is an encryption scheme created in 1982 by two scientists James Massey and Jim K. Omura. As mentioned earlier, every cryptosystem has three main components; key generation, encryption algorithm, and decryption algorithm. Here, we discuss all three components in this cryptosystem in conjunction with Elliptic Curves. This procedure works in any finite group. And is very similar to the Diffie-Hellman problem earlier.
Note: $N = \#E(F_q)$ and the message M is expressed as a point on the curve C.

Algorithm 7 Massey Omura Algorithm

```
1: procedure MasseyOmura($C, M$)                    ▷ $C : y^2 = x^3 + ax + b\ (F_q)$
2:     $A$ chooses private key $gcd(m_A, N) = 1$
3:     Compute $M_1 = m_A G$ and send to $B$
4:     $B$ chooses private key $gcd(m_B, N) = 1$
5:     Compute $M_2 = m_B(M_1)$ and send to $A$
6:     $A$ computes $M_3 = m_A^{-1} M2$ and sends to $B$
7:     $B$ computes $M_4 = m_B^{-1} M_3$
8:     return $M_4$                                 ▷ Message to Bob from Alice
9: end procedure
```

As we can see, the Massey Omura Algorithm is very similar to the ECDH method, except that, here we take the point M instead of the generator point G.

Similar to the earlier case, the time complexity for the Massey Omura Algorithm is

$$\text{T.C} = O((log\, n)^{k+1})$$

3.3.3 ElGamal Encryption Scheme

The first-ever public key encryption scheme builds over the Diffie-Hellman key exchange is the ElGamal system. And the more powerful one is the Integrated Encryption Scheme. ElGamal encryption can produce multiple ciphertexts for a single plaintext, which makes it a probabilistic encryption scheme. As a result, the size of the ciphertext is usually twice the size of the plaintext, resulting in a 1:2 expansion ratio.
In general ElGamal encryption, modular arithmetic is used to generate the ciphertext, while in ECC-based ElGamal encryption, elliptic curve arithmetic is used instead.

The algorithms for encryption and decryption are as follows:
Note: Before encryption-decryption, a public key is set up by Bob to Alice

$$Q = dG, \text{ where } 1 \leq d \leq n-1 \text{ is a private key of Bob.}$$

Algorithm 8 ElGamal Encryption

```
1: procedure ENCRYPT($C, M, G, Q$)                    ▷ Alice encrypts message
2:     Choose a random number $1 \leq k \leq n-1$
3:     Compute ciphertext $C_1 = kG$
4:     Compute ciphertext $C_2 = M + kQ$
5:     return $C_1, C_2$                              ▷ Encrypted text to Bob
6: end procedure
```

Algorithm 9 ElGamal Decryption

```
1: procedure DECRYPT($C, C_1, C_2$)
2:     Compute plain text $M = C_2 - dC_1$
3:     return $M$                                     ▷ Message to Bob from Alice
4: end procedure
```

Similar to the earlier cases, the time complexities of encryption and decryption in ElGamal are

$$\text{T.C} = O((log\, n)^{k+1}) \text{ for encryption}$$
$$\text{T.C} = O((log\, n)^{k+1}) \text{ for decryption}$$

3.3.4 Elliptic Curve Digital Signature Algorithm

ECDSA (Elliptic Curve Digital Signature Algorithm) is a digital signature algorithm based on elliptic curve cryptography. It is a variant of the Digital Signature Algorithm (DSA) that uses elliptic curve cryptography instead of finite field arithmetic. ECDSA allows a signer to sign a message using their private key, and anyone with access to their public key can verify the authenticity of the signature. Additional to the key generation, encryption, and decryption algorithms, ECDSA also uses a hash function to securely hash the message.

Note: Generally, a hash function such as SHA-256 or SHA-384 is used in ECDSA. And before signing-verifying, a public key is set up by Bob to Alice,

$$Q = dG, \text{ where } 1 \leq d \leq n-1 \text{ is a private key of Bob.}$$

Algorithm 10 ECDSA Sign

```
procedure SIGN($C, M, G, Q$)                          ▷ Alice signs the message
    $L_n$ = bit length of $n$
    Compute hash, $h = HASH(M)$
    Compute $z_n = L_n$ left most bits of $h$
    while $r \neq 0$ and $s \neq 0$ do                 ▷ $(r, s)$ to be a valid signature
        Choose private key $1 \leq k \leq n - 1$
        Compute $(x_1, y_1) = kG$
        Compute $r \equiv x_1 \; (mod\, n)$ and $s \equiv k^{-1}(z + rd) \; (mod\, n)$
    end while
    return $(r, s)$                                   ▷ Signature sent to recipient
end procedure
```

Algorithm 11 ECDSA Verify

```
procedure VERIFY($C, M, G, r, s$)                     ▷ Bob verifies signed message
    if $r, s$ do not lie in $[1, n - 1]$ then
        return Invalid signature
    end if
    $L_n$ = bit length of $n$
    Compute hash, $h = HASH(M)$                       ▷ Produces same hash
    Compute $z_n = L_n$ left most bits of $h$
    Compute $u_1 \equiv z_n s^{-1} \; (mod\, n)$
    Compute $u_2 \equiv rs^{-1} \; (mod\, n)$
    Compute $(x_1, y_1) = u_1 G + u_2 Q$
    if $(x_1, y_1) = \infty$ then
        return Invalid signature
    end if
    Compute $v \equiv x_1 \; (mod\, n)$
    if $v \neq r$ then
        return Valid Signature
    end if
    return Invalid Signature
end procedure
```

This algorithm offers many advantages for SSH, such as being faster and more secure than RSA and DSA for signing due to its smaller keys (usually 256 or 384 bits). Additionally, ECDSA is more resistant to quantum attacks as the quantum algorithms for breaking elliptic curves are less efficient than those for breaking factoring and discrete logarithms.

Time Complexity

The time complexity for ECDSA can be estimated as (N is the bit length of the prime number used in generating the curve)

$$\begin{gathered}\text{Key Generation: } O(N^2)\\ \text{Signature Generation: } O(N^3) \text{ or } O(N^4)\\ \text{Signature Verification: } O(N^3) \text{ or } O(N^4)\end{gathered}$$

The above complexity depends on the algorithm used to find the scalar multiplication and the way it is implemented. Therefore, approximating bit length as $log\, n$ where n is the prime number chosen for the field, the time complexity is

$$O((log\; n)^{k+1}); \qquad k = 2/3$$

Chapter 4

Weil Pairing

4.1 Introduction

Let E/K be an elliptic curve over the field K, and $E[n]$ represents the group of torsion points of order n in $E(\bar{K})$.

If $\text{char}(K) \nmid n$, then $E[n] = Z/nZ \bigoplus Z/nZ$. This "two-dimensionality" of the group of torsion points is what makes pairings possible (there are no interesting pairings possible of cyclic groups).

4.2 Divisors

For each point $P \in E(\bar{K})$, we define $[P]$ as the formal symbol (4).

Definition 6. *A divisor D on E is a finite linear combination of such symbols with integer coefficients, i.e., $\alpha[P] + \beta[P] = (\alpha + \beta)[P]$.*

The formal symbols generate a free Abelian group, generated by the formal symbols and divisors are elements in the group. Let this group of divisors be denoted by $\text{div}(E)$.

We define the following quantities

$$\deg\Big(\sum_j a_j[P_j]\Big) = \sum_j a_j \in Z$$
$$\text{sum}\Big(\sum_j a_j[P_j]\Big) = \sum_j a_j P_j \in E(\bar{K})$$

We will see that the divisors of degree 0 (denoted by $\text{Div}^0(E)$) form an important subgroup.

There is a surjective homomorphism with the sum function

$$\begin{aligned}&\text{sum}: \text{Div}^0(E) \to E(\bar{K})\\ &\text{sum}([P]-[\infty]) = P\end{aligned}$$

Definition 7. *A function is said to have a **zero** at a point P if it takes the value 0 at P, and it has a **pole** at P if it takes the value ∞ at P.*

Let P be a point. It can be shown that there is a function u_P, called a **uniformizer** at P, with $u(P) = 0$ and such that every function $f(x, y)$ can be written in the form

$$f = u_P^r g, \quad \text{with } r \in Z \text{ and } g(P) \neq 0, \infty$$

The existence of the uniformizer follows from the smoothness of the curve and showing that the partial derivatives are not all zero. At any finite point $P = (x_0, y_0)$ on an elliptic curve, the uniformizer u_P can be taken from the equation of a line that passes through P but is not tangent to E. A natural choice is $u_P := x - x_0 = 0$ when $y \neq 0$.

Definition 8. *The **order** of f at P is given by $ord_P(f) = r$, where $f = u_p^r g$.*

For the point $P = \infty$ and elliptic curve $E : y^2 = x^3 + Ax + B$, a uniformizer at P is $u_\infty = x/y$.

Definition 9. *If f is a function on E that is not identically 0, define the **divisor** of f to be*

$$div(f) = \sum_{P \in E(\bar{K})} ord_P(f)[P] \in Div(E) \tag{4.1}$$

Theorem 4.2.1. *Let E be an elliptic curve and let f be a function on E that is not identically 0.*

1. *f has only finitely many zeros and poles*
2. *$deg(div(f)) = 0$*
3. *If f has no zeros or poles (so $div(f) = 0$), then f is a constant*

Proof. Every rational function with coordinates from an algebraically closed field is of the form $\dfrac{f(X,Y)}{g(X,Y)}$ after homogenizing the function. Each term in the numerator and denominator have the same degree, which leads to the number of poles being equal to the number of zeros. □

The divisor of a function is said to be a **principal divisor**.

If we have a rational function $\frac{f(X,Y)}{g(X,Y)}$, then

$$\operatorname{div}\left(\frac{f(X,Y)}{g(X,Y)}\right) = \operatorname{div}(f(X,Y)) - \operatorname{div}(g(X,Y))$$

Theorem 4.2.2. *Let E be an elliptic curve and D be a divisor on E that belongs to $Div^0(E)$. There is a function f on E with $div(f) = D$ if and only if $sum(D) = \infty$.*

Proof. Suppose P_1, P_2, P_3 are three points on E that lie on the line $ax + by + c = 0$. Then the function $f(x, y) = ax + by + c$ has zeros at P_1, P_2, P_3. The divisor of $f(x, y)$ is $div(ax + by + c) = [P_1] + [P_2] + [P_3] - 3[\infty]$. The line through the points $P_3, -P_3$ is $x - x_3 = 0$. So, we have $div(x - x_3) = [P_3] + [-P_3] - 2[\infty]$.

The sum $[P_1] + [P_2]$ can be replaced by $[P_1 + P_2] + [\infty] + div(g)$ from the above argument (where $P_1 + P_2 = P_3$)

So, $\text{sum}(div(g)) = P_1 + P_2 - (P_1 + P_2) - \infty = \infty$.

Let sum of all terms in D with positive coefficients be $[P] + n_1[\infty] + div(f_1)$ and let the sum of all terms in D with positive coefficients be $[Q] + n_2[\infty] + div(f_2)$.

So, $D = [P] - [Q] + n[\infty] + div(g_1)$ for some points P, Q and some parameter n and a rational function g_1.

$\text{sum}(div(g_1)) = \infty$, it follows from $\text{sum}(div(g)) = \infty$.

$\deg(D) = 1 - 1 + n + 0 = n$

$$D = [P] - [Q] + div(g_1)$$
$$sum(D) = P - Q + sum(div(g_1)) = P - Q$$

Suppose $sum(D) = \infty$. Then $P - Q = \infty$, so $P = Q$ and $D = div(g_1)$.

Converse of the proof follows from the lemma below. □

Lemma 4.2.1. *Let $P, Q \in E(\bar{K})$ and suppose there exists a function h on E with $div(h) = [P] - [Q]$. Then $P = Q$.*

Proof. Suppose $P \neq Q$ and $div(h) = [P] - [Q]$. Then, for any constant c, the function $h - c$ has a simple pole at Q. By 4.2.1, it has exactly one zero.

Let f be any function on E. If f does not have a zero or pole at Q, then $g(x, y) = \prod_{R \in E(\bar{K})} (h(x, y) - h(R))^{ord_R(f)}$ has the same divisor as f.

Each factor has a pole at Q (of order $ord_R(f)$). Since f and g have the same divisor, the quotient f/g has no zeros or poles, and therefore is constant.

Every function on $E(\bar{K})$ is a rational function on h (in particular, x and y are rational functions of h), and the lemmas 11.5, 11.6 (1) show that such rational functions do not exist.

Hence, $P \neq Q$ is a contradiction. □

4.3 Construction of the Weil Pairing

Let n be an integer not divisible by $char(K)$ and E be an elliptic curve over K. We have already proved that $E[n] \subseteq E(K)$.

We want to construct a pairing $e_n : E[n] \times E[n] \to \mu_n$ where μ_n is the set of n^{th} roots of unity in the algebraic closure of K. Using a result we have proved, we have $E[n] \subseteq E(K) \implies \mu_n \subset K$.

1. Let $T \in E[n]$. By 4.2.2, $\exists f$ such that $div(f) = n[T] - n[\infty]$.
2. The subgroup $E[n^2]$ contains every point that satisfies $n^2P = \infty$, where $P \in E(\bar{K})$. So, $\exists T^{'} \in E[n^2]$ such that $nT^{'} = T$ since $n(nT^{'}) = nT = \infty$.
3. From 4.2.2, there exists a function g such that $div(g) = \sum\limits_{R \in E[n]} \left([T^{'} + R] - [R]\right)$
4. Let $f \circ n$ be the function that multiplies the input P with n and then applies f to it. Then $div(f \circ n) = n \left(\sum\limits_{R} [T^{'} + R]\right) - n \left(\sum\limits_{R} [R]\right) = div(g^n)$.
5. Let $S \in E[n]$ and let $P \in E(\bar{K})$. Then
   $g(P+S)^n = f(n(P+S)) = f(nP) = g(P)^n$.
6. So, $\left(\frac{g(P+S)}{g(P)}\right)^n = 1$. Therefore, $\left(\frac{g(P+S)}{g(P)}\right) \in \mu_n$.

We can observe that $\frac{g(P+S)}{g(P)}$ is independent of point P.

Definition 10. *The bilinear map $e_n : E[n] \times E[n] \to \mu_n$ is defined as follows :*
For $S, T \in E[n]$, $e_n(S,T) = \frac{g(P+S)}{g(P)}$.

4.4 MOV Attack

The goal of this attack (5) on ECDLP is to reduce the given problem to an easier discrete logarithm problem.

Algorithm 12 MOV attack on Discrete Logarithm problem

```
1: procedure MOV-ATTACK(P, Q)
2:     Input : $P \in E(F_q)$ of order $N$
3:     Output : $m$, where $Q = mP$
4:     Determine smallest integer $k$ such that $E[n] \subseteq E(F_{q^k})$.
5:     Choose a random point $R \in E(F_{q^k})$ and compute the order M of R
6:     Let $d = gcd(M, N)$, and let $R_1 = (\frac{M}{d})R$.
7:     Compute $u = e_N(P, R_1), v = e_N(Q, R_1)$
8:     Compute $l$, the discrete logarithm of $v$ to the base $u$ in $F_{q^k}$.
9:     Repeat with random points R to get a congruence of the form $k(mod\ N)$.
10: end procedure
```

4.4.1 Proof of correctness

The points of $E[n]$ have coordinates coming from the algebraic closure $\bar{F_p}$.
From finite field theory, $\bar{F_p} = \bigcup_{i=1}^{\infty} F_{p^i}$
Since there are a finite number of points in $E[n]$, we can choose k to be the largest value of all such i's. All the primitive N^{th} roots of unity are contained in F_{q^k}.
From the bilinear properties of Weil Pairing, $e_N(Q, R_1) = e_N(mP, R_1) = e_N(P, R_1)^m$.
Now, we have an easier discrete logarithm problem in F_{q^k} since $e_N(P, R_1) \in \mu_N$.

Chapter 5

Applications of ECC

5.1 Bitcoin

This section of the report has been worked upon by Anita Dash and Tata Sai Manoj.

5.1.1 Introduction

Bitcoin is a digital currency in which transactions between peers are verified and records of the transactions are maintained by a decentralized system. It uses secure methods in cryptography that does not require a central authority to have access to personal data.

The ownership of a bitcoin is determined by the wallet of the user. The wallet stores the encryption material that is required to ”access” that bitcoin, like the public address. The wallets are made secure with a unique private key that is supposed to ensure that only the user with their own private key can access their wallet.

Bitcoin works on a digital ledger called the blockchain. The blocks of data are created and introduced into the ledger (which forms a ”chain”), hence the name ”Blockchain”. The process of creating new blocks is typically through a consensus from a majority of verified nodes, which makes the blockchain highly secure against attacks with malicious intent (the attacker should have control of a high number of nodes in order to disrupt the blockchain).

A bitcoin address is a string of digits and characters that can be shared with anyone who wants to send you money. To convert a bitcoin public key to a bitcoin address, it is first hashed with a ”double-hash” function HASH160 (i.e, SHA-256 and RIPEMD160) and then encoded in Base58 with a $0x00$ prefix.

5.1.2 Secp256k1

Bitcoin employs the Elliptic Curve Digital Signature Algorithm to authorise payments by validating ownership during the exchange of cryptocurrency. The elliptic curve used in the algorithm is **Secp256k1** which is defined by the Standards for Efficient Cryptography Group (SECG).
The curve is defined by the equation

$$y^3 = x^2 + 7$$

over the field $\mathbb{F}_p$, where

$$p == 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$$

5.1.3 Secp256k1 vs Pollard's Rho Algorithm

We know that the time complexity of Pollard's Rho Algorithm is approximately $O(\sqrt{N})$. The default key length is 256 bits for secp256k1, which implies the time complexity in terms of input size in bits is around $O(2^{128})$. This might require a significant amount of resources and yet still might be infeasible.
Pollard's rho method with automorphism is a modified algorithm of Pollard Rho to speed up the time complexity (6). The key idea of the method is to find a function that runs over the equivalence classes on the automorhpism groups generated by an automorphism $\alpha : E(\mathbb{F}_p) \rightarrow E(\mathbb{F}_p)$ rather than the points and replace it in the Pollard's Rho algorithm.
However this method only speeds up the algorithm by a factor and thus secp256k1 is safe against this attack (6).

5.1.4 Secp256k1 vs Pohlig-Hellman Algorithm

The Pohlig-Hellman Algorithm reduces the problem of finding a discrete log of a point to finding discrete logs modulo the prime factors of the order of the point P. Using this algorithm along with Pollard's Rho can speed up the process to a time complexity equal to the square root of the highest prime factor order of P.
However the order of the base point in Secp256k1 is a prime number and hence there is no possibility of speeding up the algorithm using Pollard Rho. And hence this attack cannot break security systems.

5.1.5 Secp256k1 vs Transfer Methods

The key idea of these methods is to reduce efficiently an ECDLP into a discrete logarithm problem in a different group (like $\mathbb{F}_{p^m}^{\times}$). And then use index calculus to solve the DLP in sub-exponential time. Methods such as MOV Attack, Frey Ruck Attack are consider as transfer methods [(6)].
However to carry out a transfer attack attack against Secp256k1, one would have to work in a field of size approximately $10^{10^{78}}$ which is somewhat infeasible [(7)]. Thus Secp2651 is safe against transfer attacks

5.2 Attacks on ECDSA

5.2.1 PlayStation3 Console

The PlayStation 3 is a video game console developed by Sony. In the annual Chaos Communication Congress conference held in 2010 at Berliner Congress Center in Germany, a hacker group named **fail0verflow** presented a flaw in the ECDSA algorithm used by Sony.

They were able to deduce the private key used within the ECDSA, which meant that pirated and unlicensed games could be signed as "official" Sony software. The flaw in the implementation was using a pseudo-random number generator which led to different signatures being

Let (x_1, S_1) and (x_2, S_2) be two digital signatures on two different messages M_1 and M_2 using the same random number m.

$Q = mP$, and $x_1 = x_2 = x_Q$ since m is fixed. Let z_1 and z_2 be the L_n left most bits of the hashed message.

$$S_1 \equiv m^{-1}z_1(mod\ m) \quad S_2 \equiv m^{-1}z_2(mod\ m)$$
$$S_1 - S_2 \equiv m^{-1}(z_1 - z_2)(mod\ m)$$
$$m \equiv (z_1 - z_2)(S_1 - S_2)^{-1}(mod\ m)$$

So, the value of m is obtained, which leads to the value of private key as follows.

$$S_2 \equiv m^{-1}(z_2 + x_2 l)(mod\ m)$$
$$l \equiv {x_2}^{-1}(mS_2 - z_2)(mod\ m)$$

5.2.2 Bitcoin Android Security Vulnerability

Another example of a flawed implementation of ECDSA happened in the case of Bitcoin. The flaw was very similar to the case of Sony PlayStation, where the Android Java SecureRandom( ) function produced numbers that are deterministic. With a similar proof as above, the attacker can obtain the private key of the person who made the transaction, and then perform transactions impersonating that person. This vulnerability led to the theft of at least 55 **BTC**.

5.3 Twist Attack on Secp256k1

If a victim has a *Naive Implementation* of certain functions, The attacker can take advantage of this and trick the victim into revealing information about their private key. We will discuss how this is possible in detail below

5.3.1 Small Sub-Group Attack

Let the elliptic curve E have a subgroup H with a few elements compared to the order of E such that the point T on E generates H. we know that the addition operation on the elliptic curve defines a group structure on the points of an elliptic curve. Let the order of H be a prime q.
An attacker can choose a point P from subgroup H, send it to the victim as their public key and ask them to encrypt a message using their private key b. Once the victim calculates $Q = bP$ and encrypts the message using Q, the attacker can brute force by checking every element in H , and find $yP \in H$ such that $y \equiv b \pmod{q}$.
This way the attacker can get some information on the victim's private key.
If the attacker repeats the same method for a number of different subgroups whose product of orders is larger than 2^{256}, then by using Chinese Remainder Theorem the attacker can find the victim's private key.

$$b < 2^{256} \quad therefore, \quad y' \equiv b \pmod{p' > 2^{256}} \implies y' = b$$

5.3.2 Twist Attack on Secp256k1

Bitcoin uses the elliptic curve Secp256k1, whose order is a prime number.
By Lagrange's Theorem, we know that there can be no non trivial subgroups to this group.
However based on the Group Law 1.2, we can see that the addition and multiplication of points do not use the constant term B of the curve. Therefore the attacker can pick a point P from a curve that differs from Secp256k1 by a constant (also has multiple smaller subgroups) and send it to the victim.
If the victim's function that calculates addition and multiplication does not verify whether the point lies on Secp256k1, then their software will perform the calculations on the curve chosen by the attacker, and the attacker can brute force to find the victim's private key.
The attacker can find the victim's private key by slightly modifying the small subgroup attack, below we show the algorithm.

5.3.3 Algorithm

Algorithm 13 Twist attack on Secp256k1 curve

```
1: procedure TWIST-ATTACK
2:     Input: Prime $p = 2^{256} - 2^{32} - 977$, Private Key b, Secp256k1 curve E
3:     Output: Private key of Bob
4:     Consider similar elliptic curves to $E$ (namely $E_1, E_2, \cdots, E_n$) that differ by a constant term.
5:     Factorize the number of points and pick subgroups of relatively small size from each elliptic curve $(P_{11}, P_{12}, \cdots, P_{1k_1}, P_{21}, P_{22}, \cdots, P_{2k_2}, \cdots, P_{n1}, \cdots, P_{nk_n})$ and product of all sizes of the curve should be greater than $2^{256}$.
6:     For each of the subgroups, compute the generator points.
7:     If all of these points are sent to Bob, with a naive implementation of cryptography that does not verify if the point lies on the curve, the reply received will be $(Q_{11}, Q_{12}, \cdots, Q_{1k_1}, Q_{21}, Q_{22}, \cdots, Q_{2k_2}, \cdots, Q_{n1}, \cdots, Q_{nk_n})$
8:     Using the Pollard-Rho algorithm for these small subgroups, we can compute the discrete logarithms for all points received.
9:     The Chinese Remainder Theorem is used to construct the private key from the discrete logarithms and the subgroup sizes.
10: end procedure
```

5.3.4 Time Analysis

For the following analysis, we are using the same twist curves and subgroups. Recovering the private key from the subgroups and the discrete logarithms is computationally the slowest step, which is highlighted in the table below. All of the computations were done using the software Sagemath 9.0 and the codes can be found here.

Time Complexity			
Private Key (Hex)	Bit Size	Steps: $7-9$	Total Runtime
0xc3549423	32	387 s	433.2274 s
0xf837bc8c4461ccd7	64	765 s	809.7708 s
0x6a9d9fd679c147a78cbc6857dec02e76	128	877 s	925.3762 s

5.4 Conclusion

Mathematically Secp2561 is a good choice for ECC. it is safe and can resist numerous possible attacks like Pollard-Rho, Modified Pollard-Rho and even transfer attacks like MOV attacks are infeasible. However, due to implementation flaws, the security system may be vulnerable to specific attacks like the twist attack.

5.5 Secure Communication

This section of the report has been worked upon by Kethari Narasimha Vardhan.

5.5.1 Introduction

The current world entirely works digitally. It is very important to secure your data and communicate securely. Secure communication refers to the process of encrypting data exchanged between two parties to protect it from unauthorized access or interception. ECC can be used for secure communication in a number of ways. One of the most common ways is through the use of the Transport Layer Security (TLS) protocol and Secure Sockets Layer (SSL) protocol.

The main difference between SSL and TLS is that TLS provides better security and supports stronger encryption algorithms. The major role of Elliptic Curves in TLS is for key exchange and Digital Signatures. The latest version, SSL 3.0, was depreciated in 2015 due to serious vulnerabilities such as *replay attacks* and *man-in-the-middle attacks.*

Here, we see how ECC is used in TLS Protocol.

5.5.2 Transport Layer Security (TLS)

The two major uses of ECC in TLS are the key exchange and digital signatures.

Key Exchange: TLS uses a key exchange mechanism to establish a shared secret key between the client and server, which is used to encrypt and decrypt data exchanged between them. ECC can be used for the key exchange mechanism in TLS through the Elliptic Curve Diffie-Hellman (ECDH) key exchange protocol, which is a variant of the traditional Diffie-Hellman protocol that uses elliptic curves.
Digital Signatures: TLS also uses digital signatures to authenticate the identity of servers and clients and to protect against man-in-the-middle attacks. ECC can be used for digital signatures in TLS through the Elliptic Curve Digital Signature Algorithm (ECDSA), which is a variant of the Digital Signature Algorithm (DSA) that uses elliptic curves.

For the **key exchange**, the algorithm of ECDH in TLS works as follows

1. The client and server agree on a set of elliptic curve parameters, such as the curve type and size, to use for the key exchange.

2. The server generates an ECC key pair consisting of a private key and a corresponding public key, and sends the public key to the client.

3. The client generates its own ECC key pair, and uses the server's public key to compute a shared secret key.

4. The client sends its own public key to the server, along with some additional information about the key exchange, encrypted using the shared secret key.

5. The server uses its private key to decrypt the client's message and compute the shared secret key.

6. Once the shared key has been established, it can be used to encrypt and decrypt data exchanged between the client and the server.

For the **digital signatures**, the algorithm of ECDSA in TLS works as follows

1. The server generates an ECC key pair, consisting of a private key and a corresponding public key.

2. The server sends its public key to a certificate authority (CA) to obtain a TLS certificate. The TLS certificate includes the server's public key and a digital signature generated using the CA's private key.

3. When a client connects to the server, the server sends its TLS certificate to the client.

4. The client uses the CA's public key to verify the digital signature in the TLS certificate and to authenticate the server's identity.

5. Once the digital signature is verified by the client, it can be confident about the legitimacy and thus a secure connection has been established.

The most common curves used SSL/TLS Protocols are NIST curves P-256 (also known as $secp256r1$) and P-384 (also known as $secp384r1$).

5.5.3 Example

Let us take the standard example of the Wikipedia page, *https://wikipedia.org/*

The TLS certificate for this page is generally issued by Digicert Inc from the US. This Certificate uses Elliptic Curve Public Key Algorithm with Signature Algorithm being ECDSA. And the curve used is $secp256r1$, its parameters are;

- $y^2 = x^3 - 3x + b \ (\mathbb{F}_p)$
- $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$
- $b = 2^{255} + 2^{119}$

The major components in this TLS Certificate are

- Common Name (CN): This is the fully qualified domain name (FQDN) for which the certificate is issued. The CN must match the domain name that users enter in their web browser in order for the certificate to be trusted. **E.g. *.wikipedia.org**
- Validity Period: This specifies the length of time that the certificate is valid. Most certificates are issued for a period of 1-2 years, after which they must be renewed. **E.g. Saturday, 18 November 2023 at 05:29:59 IST**
- Public Key: The certificate includes a public key that is used to encrypt data sent to the website. The private key, which is kept secret by the website owner, is used to decrypt the data. **E.g. Elliptic Curve Public Key (secp256r1)**
- Issuer: The certificate issuer is the organization that verifies the identity of the website owner and issues the certificate. The issuer's identity is also included in the certificate. **E.g. DigiCert Inc, USA**
- Signature Algorithm: This is the algorithm used to sign the certificate and verify its authenticity. Common signature algorithms include SHA-256 and SHA-384. **E.g. ECDSA with SHA-384**
- Key Usage: This specifies how the public key in the certificate can be used, such as for encryption, digital signatures, or key agreement. **E.g. Encrypt, Verify, and Derive**

Note: Some of the results in the components of the TLS certificate might vary as the server generates different public-private key pairs for different clients.

References

[1] L. C. Washington. Elliptic Curves : Number Theory and Cryptography. 2nd edition. Chapman and Hall, 2008. 2, 5, 6, 10, 24

[2] P. Flajolet and A. M. Odlyzko. Random Mapping Statistics. 2001. 12

[3] R. A. Mollin. An Introduction to Cryptography. 2007. 13

[4] J. H. Silverman. The Arithmetic of Elliptic Curves. 2009. 21

[5] A. J. Menezes, T. Okamoto, and S. A. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field, IEEE Trans. Inform. Theory. 1993. 25

[6] `https://blog.coinfabrik.com/wp-content/uploads/2016/06/ECDSA-Security-in-Bitcoin-and-Ethereum-a-Research-Survey.pdf`. 27, 28

[7] `http://koclab.cs.ucsb.edu/teaching/ecc/project/2015Projects/Schmid.pdf`. 28

[8] `https://www.researchgate.net/publication/322558426_RSA_and_ECC_A_comparative_analysis`.

[9] `http://www.umsl.edu/~siegelj/information_theory/projects/EllipticCurveEncyiption.pdf`.

[10] `https://www.researchgate.net/publication/265945652_A_Discussion_on_Elliptic_Curve_Cryptography_and_Its_Applications`.

[11] `https://csrc.nist.gov/CSRC/media/Events/Key-Management-Workshop-2001/documents/p1363.pdf`.

[12] `https://www.ripublication.com/ijaer17/ijaerv12n19_140.pdf`.

[13] `https://ijcert.org/ems/ijcert_papers/V3I1101.pdf`.

[14] https://csrc.nist.gov/csrc/media/events/workshop-on-elliptic-curve-cryptography-standards/documents/papers/session6-adalier-mehmet.pdf?ref=blog.cloudflare.com.

[15] https://koclab.cs.ucsb.edu/teaching/ecc/project/2015Projects/Levy.pdf.

[16] https://koclab.cs.ucsb.edu/teaching/ecc/project/2015Projects/Vasudevan.pdf.

[17] https://gist.github.com/tscholl2/b5d7a64fe9d283ee2b2bfe46588237a7.

[18] http://koclab.cs.ucsb.edu/teaching/ecc/project/2015Projects/Sommerseth+Hoeiland.pdf.

[19] https://journals.riverpublishers.com/index.php/JCSANDM/article/view/15085.

[20] https://learn.saylor.org/mod/book/tool/print/index.php?id=36353.

[21] https://github.com/bitcoinbook/bitcoinbook/blob/develop/ch04.asciidoc.

[22] https://en.bitcoin.it/wiki/Secp256k1.

[23] https://lewismcombes.github.io/downloads/research/mathematics_of_bitcoin_ecdsa_lmc.pdf.

[24] https://bitcoin.org/en/alert/2013-08-11-android.

[25] https://github.com/christianlundkvist/blog/blob/master/2020_05_26_secp256k1_twist_attacks/secp256k1_twist_attacks.md.

[26] https://crypto.stanford.edu/pbc/notes/elliptic/weil.html.

[27] https://math.mit.edu/classes/18.783/2017/LectureNotes24.pdf.