CHAPTER 3
MATHEMATICAL ANALYSIS OF ECC

3.1 INTRODUCTION

For working with EC based crypto system, a mathematical analysis will enable clear understanding and proper implementation. This chapter deals with the theoretical and mathematical background of EC. Before beginning with the description of EC, a brief introduction to some important and necessary concepts about groups and finite fields is detailed.

3.1.1 Groups

A set $C$ is called an abelian group $(C, \cdot)$ with a binary operation $\cdot : C \times C \rightarrow C$ if it satisfies the following properties (Hankerson et al 2004):

- **Associativity:** $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ for all $a, b, c \in C$.
- **Commutativity:** $a \cdot b = b \cdot a$ for all $a, b \in C$.
- **Identity:** There exists an element $i \in C$ such that $a \cdot i = i \cdot a = a$ for all $a \in C$.
- **Inverse:** For each $a \in C$, there exists an element $b \in C$, called the inverse of $a$, such that $a \cdot b = b \cdot a = i$.

The group is called additive or multiplicative if the binary operation is called addition (+) or multiplication (\cdot), respectively. For the first case, the identity element is usually denoted by 0, and the additive inverse of an
element $a$, by $-a$. For the second case, the identity element is usually denoted by 1, and the multiplicative inverse of an element $a$, by $a^{-1}$.

Let $(C, \cdot)$ be a group and $B$ be a subset of $C$. The structure $(B, \circ)$ is said to be a subgroup of $(C, \cdot)$, if $\circ$ is the restriction of $\cdot$ to $B \times B$ and $(B, \circ)$ is a group. The group is finite if $C$ is a finite set, in which case the number of elements in $C$ is called the order of $C$ and it is denoted as $|C|$. Given a finite multiplicative group $C$, the order of an element $a \in C$ is the smallest positive integer $m$ such that $a^m = 1$. Such an $m$ exists for every element in a finite multiplicative group, as follows from the next theorem and its corollary.

**Theorem 3.1**: Let $C$ be a finite multiplicative group of order $n$. If the order of an element $a \in C$ is $m$, then $a^k = 1$ if and only if $m|k$.

**Proof**: If $k = mq$, then $a^k = (a^m)^q = 1$. For the converse, let $k = mq + r$, $0 \leq r < m$. Then $a^r = a^k(a^{-1})^{mq} = 1$. Therefore, it follows the minimality of $m$ that $r$ must be 0.

**Corollary 3.1**: If $C$ is a finite multiplicative group of order $n$, then

- for every element $a \in C$, $a^n = 1$.
- the order of any element of $C$ divides $|C|$.

If $a \in C$ is of order $m$, then $B = \{a^k \mid k \in \mathbb{Z}\}$ is a subgroup of $C$ of order $m$. If $C$ has an element $a$ of order $n = |C|$, then $C = \{a^k \mid k \in \mathbb{Z}\}$ and $C$ is called cyclic and $a$ is called a generator of $C$. The set $Z_n = \{0, 1, 2, ..., n\}$ is a cyclic group of order $n$ under addition modulo $n$, i.e. $a + b \equiv r \mod n$, where $r < n$ (r is the remainder when $a + b$ is divided by $n$).
3.1.2 Finite Fields

The concept of finite fields is briefed through the definition of the finite prime field $F_p$ that is used throughout this work. First, consider the finite group with order prime $p$, $F_p$: \{0, 1, \ldots, p - 1\}. Then, $(F_p, +)$ is an additive finite group of order $p$ and additive identity 0, and $(F_p^*, \cdot)$ is a multiplicative finite group of order $p$ and multiplicative identity 1, where $F_p^*$ denotes the nonzero elements in $F_p$. The triple $(F_p, +, \cdot)$, simply known as $F_p$ or prime field, is a finite field.

As an extension of the definition of groups, the prime field $F_p$ is finite since it contains a finite number of elements given by $p$, which also represents the order of the field. In a more general sense, given a field $F_q$ of order $q$, it is said to be a finite field if and only if its order is a prime power $q = p^m$. In particular, it is seen that $F_q$ is a prime field if $m = 1$. Later, it is noted that all the points belonging to an EC over such finite field $F_p$ will be used to implement the EC based cryptosystem.

3.2 ELLIPTIC CURVES OVER $F_p$

Elliptic curves have a rich and glorious history. The problem of computing the arc length of an ellipse gave rise to elliptic functions that satisfy cubic equations, hence plane cubic curves are called elliptic curves. Elliptic curves link number theory, algebraic geometry and complex analysis, and have applications to factorization of integers, cryptography and coding theory. There are many famous unsolved problems and conjectures involving elliptic curves and they were also crucial in Andrew Wile's famous proof of Fermat's last theorem.
An elliptic curve $E$ over a field $K$ denoted by $E(K)$ is defined by the general Weierstrass equation (Hankerson et al 2004):

$$E : y^2 + a_1 xy + a_3 y = x^3 + a_2 x^2 + a_4 x + a_6$$

(3.1)

where $a_1, a_2, a_3, a_4, a_6 \in K$ and $\Delta \neq 0$, which is the discriminant of $E$.

The previous condition guarantees that there do not exist more than one tangent line for a given point on the curve, i.e., the curve is “smooth”. The set of pairs $(x, y)$ that solves the equation (3.1) and the point at infinity $O$, which is the identity for the group law, form an abelian group $(E(K), +)$ with binary operation denoted by addition. This group of points is used to implement the elliptic curve cryptosystem.

ECC is defined over different finite fields $K$. Most important finite fields used to implement this cryptosystem have been binary, prime and extension fields. The present thesis works with a prime field, denoted by $F_p$, where $p$ is a large prime and also represents the number of elements of the field. In this case, the general Weierstrass equation simplifies to the following (Hankerson et al 2004):

$$E : y^2 = x^3 + ax + b$$

(3.2)

where $a, b$ are constants and $\Delta = 4a^3 + 27b^2 \neq 0$.

Consequently, the set of pairs $(x, y)$ that solves the equation (3.2), where $x, y \in F_p$, and the point at infinity $O$ form an abelian group $(E(F_p), +)$, which ultimately contains all the possible elements for computations on ECC over prime fields:

$$(E(F_p), +) = \{(x, y) \in F_p \times F_p : y^2 - x^3 - ax - b = 0\} \cup \{O\}$$

(3.3)
Hence, it is remarked without further proofs at this point that defined group \((E(F_p), +)\) satisfies properties corresponding to abelian groups.

- Addition is commutative and associative for all points \(P = (x_1, y_1) \in E(F_p)\).
- The additive identity is denoted by \(O\), such that \(P + O = O + P = P\) for all \(P \in E(F_p)\).
- Every \(P = (x_1, y_1) \in E(F_p)\) has an additive inverse denoted by \(-P\). Later, it will be seen that \(-P = (x_1, -y_1)\).

The following section, succinctly describes the arithmetic layers that constitute the computation of the scalar multiplication.

### 3.3 ELLIPTIC CURVE ARITHMETIC

Cryptographic mechanisms based on elliptic curves depend on arithmetic involving the points of the curve. Curve arithmetic is defined in terms of underlying field operations, the efficiency of which is essential. Efficient curve operations are likewise crucial to performance. The mathematical hierarchy of the ECC scalar multiplication consists of three levels: finite field arithmetic, point arithmetic and scalar arithmetic as shown in Figure 3.1.

#### 3.3.1 Level 1: Finite Field Arithmetic

Basic curve operations in ECC over prime fields are performed using field operations. The finite field arithmetic operations are

- Addition: Given \(a, b \in \mathbb{Z}_p\), \((a + b) \mod p = r\), where \(r\) is the remainder of dividing \(a + b\) by \(p\), and \(0 \leq r \leq p - 1\).
Subtraction: Given \( a, b \in F_p \), \((a - b) \mod p = r\), where \( r \) is the remainder of dividing \( a - b \) by \( p \), and \( 0 \leq r \leq p - 1 \). This operation is commonly replaced by an addition performed on \( a \) and \((-b)\), given that the negative of any element is easily obtained.

Multiplication: Given \( a, b \in F_p \), \((a \cdot b) \mod p = r\), where \( r \) is the remainder of dividing \( a \cdot b \) by \( p \), and \( 0 \leq r \leq p - 1 \).

Squaring: Given \( a \in F_p \), \((a^2) \mod p = r\), where \( r \) is the remainder of dividing \( a^2 \) by \( p \), and \( 0 \leq r \leq p - 1 \).

Inversion: given \( a \), a non-zero element in \( F_p \), \((a^{-1}) \mod p = r\), is the unique integer \( r \in F_p \), for which \((a \cdot r) \mod p = 1\).

**Example 3.1:** Given the elements of prime field \( F_{29} \) are \( \{0, 1, 2, \ldots, 28\} \), examples of arithmetic operations over such finite field are:

- \( 17 + 20 = 37 \equiv 8 \mod 29 \).
- \( 17 - 20 = -3 \equiv 26 \mod 29 \).
- \( 17 \cdot 20 = 340 \equiv 21 \mod 29 \).
\[ 17^2 = 289 \equiv 28 \mod 29. \]
\[ 17^{-1} \equiv 12 \mod 29, \text{ since } 17 \cdot 12 = 204 \equiv 1 \mod 29. \]

### 3.3.2 Level 2: Point Arithmetic

Scalar multiplication directly depends on operations over points on the elliptic curve \( E \). In general, traditional methods to compute the scalar multiplication on EC is repeated point addition. To improve the efficient computation of the EC, scalar multiplication rely on the execution of a given sequence of point doubling (\( 2P \)) and point addition (\( P + Q \)) operations, where \( P \) and \( Q \) are points on the elliptic curve \( E \). Formulae to compute the previous elementary point operations are derived according to what is best known as group law.

Elementary point operations are geometrically described in order to understand the way in which point formulae are derived. The following description is based on the natural representation of points using \( x \) and \( y \) coordinates, which is called in the context of ECC affine coordinates representation. Given an elliptic curve over a field \( K, E(K) \), the resultant point \( P + Q = (x_3, y_3) \) of adding two points \( P = (x_1, y_1) \) and \( Q = (x_2, y_2) \) in \( E(K) \) is the reflection about the \( x \)-axis of the point that is intersected by the line crossing \( P \) and \( Q \). Figure 3.2 (a) permits to visualize such operation.

The exception to the previous procedure is the case where the points to be added have identical \( x \)-coordinate. Given equation (3.2), for any \( x = x_1 \), one has two solutions \( y = \pm y_1 \), and consequently, two points with the same \( x \)-coordinate, i.e., \( (x_1, y_1) \) and \( (x_1, -y_1) \). The latter is simply solved by using the identity of the group law, namely the point at infinity \( O \), which can be geometrically defined as the point “lying far out on the \( y \)-axis such that any
line $x = c$, for some constant $c$, parallel to the $y$-axis passes through it” (Avanzi et al 2005).

Thus, the line crossing $(x_1, y_1)$ and $(x_1, -y_1)$ obviously intersects the curve in the point at infinity $O$. Following the same definition, the reflection of $O$ about the $x$-axis gives again the point at infinity $O$, so that $(x_1, y_1) + (x_1, -y_1) = O$. As a consequence, it is defined the negative of $P = (x_1, y_1)$ as $-P = (x_1, -y_1)$.

By following the previous geometric description, formula for the point addition in affine coordinates has been derived and is described in the following (Hankerson et al 2004). Let $E(F_p)$ be an elliptic curve over the prime field $F_p$, where $p > 3$. Given two points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ on $E(F_p)$, where $P \neq \pm Q$, the addition $P + Q = (x_3, y_3)$ is obtained as follows:

**Figure 3.2 Geometric addition and doubling of elliptic curve points**
\[ x_3 = \lambda^2 - x_1 - x_2, \quad y_3 = \lambda (x_1 - x_3) - y_1 \]  

(3.4)

where \( \lambda = \frac{y_2 - y_1}{x_2 - x_1} \)

Similarly, doubling of a point can be explained by its geometric description. Given an elliptic curve over a field \( K, E(K) \), the resultant point \( 2P = (x_3, y_3) \) of doubling the point \( P = (x_1, y_1) \) in \( E(K) \) is the reflection about the \( x \)-axis of the point that is intersected by \( y \) the tangent line of \( P \). Figure 3.2 (b) permits to visualize such operation.

Formula for the point doubling in affine coordinates can be easily derived from the previous geometric description. It is described in the following (Hankerson et al 2004). Let \( E(F_p) \) be an elliptic curve over the prime field \( F_p \), where \( p > 3 \). Given a point \( P = (x_1, y_1) \) on \( E(F_p) \), where \( P \neq -P \), the doubling \( 2P = (x_3, y_3) \) is obtained as follows:

\[ x_3 = \lambda^2 - 2x_1, \quad y_3 = \lambda (x_1 - x_3) - y_1 \]  

(3.5)

where \( \lambda = \frac{3x_1^2 + a}{2y_1} \)

**Remark 3.1:** \( O \) serves as the additive identity ("zero element") of the group of points.

### 3.3.3 Level 3: Scalar Arithmetic

This mathematical level deals with the efficient computation of scalar multiplication using point operations explained in the previous section. Let \( E \) be the elliptic curve over the finite field \( F_p \). The scalar multiplication operation in ECC is represented as follows:
\[ Q = kP \]  \hspace{1cm} (3.6)

where \( P \) and \( Q \) are points in \( E(F_p) \) of order \( q \), and \( k \) is the secret scalar.

For an \( n \)-bit scalar multiplication \( kP \) where \( n \) is the bit length of the prime \( p \) corresponding to the prime field \( F_p \), it is assumed that \( P \) is of order \( h.q \) (\( q \) prime and \( h << q \)) and \( n \approx \log_2 q \) (i.e., \( p \approx q \)) (Hankerson et al 2004). If \( k \) is a scalar randomly chosen in the range \([1, q - 1]\), then the average length of \( k \) is \( l \approx n - 1 \). This is referred to as density or hamming weight to the number of nonzero elements in a given integer representation. In particular, for scalar multiplication, the latter is directly translated to the number of required point additions to compute \( kP \) using such representation.

The binary method of scalar multiplication is based on the binary expansion of the scalar \( k \) using \( \{0, 1\} \). Given a binary representation of \( k \), the scalar multiplication can be computed by scanning the bits of \( k \) from left to right, as shown in algorithm ‘computeScalarMul’.

**Algorithm computeScalarMul( )**

// Input : \( k = (k_{l-1} \ldots k_0)_2, P \in E(F_p) \)
// Output : \( kP \)

\[
\begin{align*}
Q &= O \\
\text{For } i = l - 1 \text{ downto } 0 \text{ do} \\
& \quad \quad \{ \\
& \quad \quad \quad Q = 2Q \\
& \quad \quad \quad \text{If } k_i = 1 \text{ then } Q = Q + P \\
& \quad \quad \} \\
\text{Return } (Q)
\end{align*}
\]

The average number of doublings \( D \) and additions \( A \) using algorithm ‘computeScalarMul’ is \( l \approx n - 1 \) and \( l/2 \approx n/2 \), respectively, as \( k \)
tends to infinity. Thus, the cost of the binary method is approximately as follows:

\[(n-1)D + \left(\frac{n}{2}\right)A\]  \hspace{1cm} (3.7)

### 3.4 Elliptic Curve Discrete Logarithm Problem

The ECDLP is defined as the problem of determining scalar \( k \), given \( P \) and \( Q \) as specified in equation (3.6). Security of systems based on ECC relies on the hardness of this problem. In general, ECDLP has proven to be harder than other recognized problems such as the IFP and the DLP, which are the foundations of RSA and DH cryptosystems, respectively.

**Definition 3.1** (Hankerson et al 2004) Given an algorithm with input \( n \), where \( n \) is an integer with size \( l \approx \log_2 n \), its running time is:

\[L_n[a, c] = O\left(\epsilon^{c-O(1)}(\log_2 n)^a (\log_2 \log_2 n)^{a-\epsilon}\right)\]  \hspace{1cm} (3.8)

where \( c > 0 \) and \( 0 \leq a \leq 1 \) are constants.

The running time of the equation (3.8) is said to be polynomial in \( l(O(\epsilon)) \) if \( a = 0 \), exponential in \( l \) if \( a = 1 \) and sub-exponential in \( l \) if \( 0 < a < 1 \). In particular, there exists a sub-exponential attack called NFS to solve the IFP and DLP in the following expected running time:

\[L_n\left[\frac{1}{3}, 1.923\right]\]  \hspace{1cm} (3.9)

In contrast, the fastest known method to solve ECDLP is Pollard’s rho, which is exponential with the following expected running time:

\[\sqrt{\pi q} \quad 2\]  \hspace{1cm} (3.10)
where $q$ is the order of $P$ and $Q$ in equation (3.6). Therefore, it is expected that smaller key sizes are required for ECC for a given security level.

Table 3.1 shows the equivalent key sizes for EC, RSA and DL cryptosystems for an equivalent level of security (Hankerson et al 2004). Estimates are based on the time to run the fastest algorithms i.e., Pollard’s rho and NFS that solve each problem. Security level is expressed by the key size in bits, meaning that for a key (scalar $k$) of size $l$, one would require $2^l$ steps to break the cryptosystem.

**Table 3.1 Key sizes for EC, RSA and DL based cryptosystems for equivalent security levels**

<table>
<thead>
<tr>
<th>Cryptosystem</th>
<th>Key sizes (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECC</td>
<td>160</td>
</tr>
<tr>
<td>RSA / DL</td>
<td>1024</td>
</tr>
</tbody>
</table>

From Table 3.1, it is observed that ECC requires much smaller keys. This is directly translated to faster computations and reduced memory requirements, which make this cryptosystem ideal for devices with constrained resources such as smartcards, cell phones, PDAs, laptops, and many others.

### 3.4.1 Algorithms known for the ECDLP

This section briefly overviews the algorithms known for the ECDLP. All of these algorithms take fully exponential time.
- **Naive exhaustive search**

  In this method, the successive multiples of $P$ are computed as $P$, $2P$, $3P$, $4P$, ... until $Q$ is obtained. This method can take up to $q$ steps in the worst case where $q$ is the order of $P$ and $Q$.

- **Baby Step Giant Step algorithm**

  This algorithm is a time memory trade off of the method of exhaustive search. It requires storage for about $\sqrt{q}$ points, and its running time is roughly $\sqrt{q}$ steps in the worst case.

- **Pollard’s rho algorithm**

  This algorithm as per Pollard (1978) is a randomized version of the BSGS algorithm. It has roughly the same expected running time $\sqrt{\pi q/2}$ steps as the BSGS algorithm, but is superior in that it requires a negligible amount of storage. Wiener and Zuccherato (1999) showed how Pollard’s rho algorithm can be speeded up by a factor of $\sqrt{2}$. Thus the expected running time of Pollard’s rho method with this speedup is $\sqrt{\pi q/2}$ steps.

- **Pohlig Hellman algorithm**

  This algorithm developed by Pohlig and Hellman (1978), exploits the factorization of $q$, the order of the point $P$. The algorithm reduces the problem of recovering $l$ to the problem of recovering $l$ modulo each of the prime factors of $q$. The desired number $l$ can then be recovered by using the CRT. To construct the most difficult instance of the ECDLP, one must select an EC whose order is divisible by a large prime $q$. Preferably, this order should be a prime or almost a prime.
• **Distributed version of Pollard’s rho algorithm**

Oorschot et al (1999) showed how Pollard’s rho algorithm can be parallelized so that when the algorithm is run in parallel on \( m \) processors, the expected running time of the algorithm is roughly \( \frac{\sqrt{pq}}{(2m)} \) steps. That is, using \( m \) processors results in an \( m \)-fold speedup. This distributed version of Pollard’s rho algorithm is the fastest general purpose algorithm known for the ECDLP.

• **Pollard’s lambda method**

This is another randomized algorithm of Pollard (1978). Like Pollard’s rho method, the lambda method can also be parallelized with a linear speedup. The parallelized lambda method is slightly slower than the parallelized rho method Oorschot et al (1999). The lambda method is, however, faster in situations when the logarithm being sought is known to lie in a subinterval \([0, b]\) of \([0, q - 1]\), where \( b < 0.39q \) Oorschot et al (1999).

• **Multiple Algorithms**

Silverman and Stapleton (1997) observed that if a single instance of the ECDLP for a given elliptic curve \( E \) and a base point \( G \) is solved using Pollard’s rho method, then the work done in solving this instance can be used to speed up the solution of other instance of the ECDLP for the same curve \( E \) and base point \( G \). More precisely, solving \( k \) instances of the ECDLP for the same curve \( E \) and base point \( G \) takes only \( \sqrt{k} \) as much work as it does to solve one instance of the ECDLP. This analysis, however, does not take into account storage requirements. Concerns that successive logarithms become easier can be addressed by ensuring that the elliptic parameters are chosen so that the first instance is infeasible to solve.
3.5 ELLIPTIC CURVE DOMAIN PARAMETERS

The operation of each of the public key cryptographic schemes described in this thesis involves arithmetic operations on an EC over a finite field determined by some elliptic curve domain parameters. This section briefs the provision of elliptic curve domain parameters. It describes what elliptic curve domain parameters are, how they should be generated and how they should be validated.

Domain parameters for an elliptic curve scheme precisely specify an elliptic curve $E$ defined over a finite field $F_p$, a base point $G \in E(F_p)$, and its order $n$. The parameters should be chosen so that the ECDLP is resistant to all known attacks. There may also be other constraints for security or implementation reasons. Typically, domain parameters are shared by a group of entities, however, in some applications they may be specific to each user.

**Definition 3.2:** Domain parameters comprised of $D = (q, \text{FR}, S, a, b, G, n, h)$ where

- $q$ - Field order.
- FR - Field Representation used for the elements of $F_p$.
- $S$ - Seed if the EC was randomly generated using algorithm ‘generateRandomEC’.
- $a, b$ - Two coefficients that define the equation of the EC over $F_p$.
- $G$ - Base point has field elements $(x_G, y_G) \in E(F_p)$ in affine coordinates.
- $n$ - Order of $G$.
- $h$ - Cofactor as $\#E(F_p) / n$.

In order to avoid the Pohlig Hellman attack and Pollard’s rho attack on the ECDLP, it is necessary that $\#E(F_p)$ be divisible by a sufficiently large
prime $n$. At a minimum, one should have $n > 2^{160}$. Having fixed an underlying field $F_p$, maximum resistance to the Pohlig Hellman and Pollard’s rho attacks is attained by selecting $E$ so that $\#E(F_p)$ is prime or almost prime, that is, $\#E(F_p) = hn$ where $n$ is prime and $h$ is small (e.g., $h = 1, 2, 3$ or 4).

### 3.5.1 Domain Parameter Generation and Validation

The algorithm ‘generateDomainParameter’ is one of the methods to generate cryptographically secure EC domain parameters over $F_p$ wherein all the security constraints are satisfied. A set of EC domain parameters over $F_p$ can be explicitly validated using algorithm ‘validateDomainParameter’. The validation process proves that the EC in question has the claimed order and resists all known attacks on the ECDLP, and that the base point has the claimed order. An entity that uses ECs generated by un-trusted software or parties can use validation to be assured that the curves are cryptographically secure. Sample sets of EC domain parameters over $F_p$ are provided in Appendix 1.

**Algorithm generateDomainParameter( )**

// **Input** : A field order $q$, a field representation FR for $F_p$, security level $L$ satisfying $160 \leq L \leq \lfloor \log_2 q \rfloor$ and $2^L \geq 4\sqrt{q}$

// **Output** : Domain parameters $D = (q, FR, S, a, b, G, n, h)$
{

step1: Select $a, b \in F_p$ to determine the EC $E(F_p)$ using algorithm ‘generateRandomEC’ and $S$ be the seed returned.

Compute $N = \#E(F_p)$

Check if $N$ is divisible by a large prime $n$ satisfying $n > 2^L$, else go to step 1

For $k$ from 1 to 20 do
\{ 
    \text{If } n \text{ is divisible by } q^k - 1 \text{ then go to step 1} 
\}

Verify that \( n \neq q \). If not, then go to step 1

Let \( h = N / n \)

Do
\{ 
    \text{Select an arbitrary point } G' \in E(F_p) \text{ and set } G = hG' 
\} \text{ while } (G \neq \infty)

Return \((q, \text{FR}, S, a, b, G, n, h)\)

\textbf{Algorithm validateDomainParameter( )}

\textbf{// Input} \quad \text{Domain parameters } D = (q, \text{FR}, S, a, b, G, n, h)

\textbf{// Output} \quad \text{Acceptance or rejection of the validity of } D

\{ 
    \text{Check that } q \text{ is a prime power } (q = p^m \text{ where } p \text{ is prime and } m \geq 1) 
    \text{If } p = 2 \text{ then verify that } m \text{ is prime} 
    \text{Verify that FR is a valid field representation} 
    \text{Check that } a, b, x_G, y_G \in F_p 
    \text{Verify that } a \text{ and } b \text{ define an EC over } F_p 
    \text{If the EC was randomly generated then} 
    \{ 
        \text{Verify that } S \text{ is a bit string of length at least } l \text{ bits, where } l \text{ is the} 
        \text{bit length of the hash function } H 
        \text{Use algorithm ‘verifyRandomEC’ to verify that } a \text{ and } b \text{ were} 
        \text{properly derived from } S 
    \}
    \text{Verify that } G \neq \infty 
\}
Verify that $G$ satisfies the EC equation defined by $a, b$
Check that $n$ is prime, that $n > 2^{160}$, and that $n > 4\sqrt{q}$
Verify that $nG = \infty$
Compute $h' = \lfloor (\sqrt[n]{q} +1)^2 / n \rfloor$ and verify that $h = h'$
For $k$ from 1 to 20 do
{} Verify that $n$ does not divide $q^k - 1$
}
Verify that $n \neq q$
If any verification fails then
{} Return “Invalid”
Else
{} Return “Valid”

3.5.2 Generating Elliptic Curves Verifiably at Random

Algorithm ‘generateRandomEC’ is specifications for generating random elliptic curves over a prime field $F_p$. The corresponding verification procedures are presented as algorithm ‘verifyRandomEC’.

Algorithm generateRandomEC( )

// Input : A prime $p > 3$, and an $l$-bit hash function $H$
// Output : A seed $S$, and $a, b \in F_p$, defining an EC $E : y^2 = x^3 + ax + b$
{
  Set $t = \lceil \log_2 p \rceil$, $s = \lfloor (t - 1) / l \rfloor$, $v = t - sl$
step 2: Select an arbitrary bit string $S$ of length $g \geq l$ bits
  Compute $h = H(S)$
Let $r_0$ be the bit string of length $v$ bits obtained by taking the $v$ rightmost bits of $h$
Let $R_0$ be the bit string obtained by setting the leftmost bit of $r_0$ to 0
Let $z$ be the integer whose binary representation is $S$
For $i$ from 1 to $s$ do
{
    Let $s_i$ be the $g$-bit binary representation of the integer $(z + i) \mod 2^g$
    Compute $R_i = H(s_i)$
}
Let $R = R_0 \ || \ R_1 \ || \ \cdots \ || \ R_s$
Let $r$ be the integer whose binary representation is $R$
If $(r = 0$ or $4r + 27 \equiv 0 \ (\mod p))$ then go to step 2
Select arbitrary $a, b \in F_p$, not both 0, such that $r \cdot b^2 \equiv a^3 \ (\mod p)$
Return $(S, a, b)$

Algorithm verifyRandomEC() 
// Input : Prime $p > 3$, $l$-bit hash function $H$, seed $S$ of bit length $g \geq l$, and $a, b \in F_p$ defining an EC $E : y^2 = x^3 + ax + b$
// Output : Acceptance or rejection that $E$ was generated using algorithm ‘generateRandomEC’ 
{ 
    Set $t = \lceil \log_2 p \rceil$, $s = \lfloor (t - 1) / l \rfloor$, $v = t - sl$
    Compute $h = H(S)$
    Let $r_0$ be the bit string of length $v$ bits obtained by taking the rightmost bits of $h$
    Let $R_0$ be the bit string obtained by setting the leftmost bit of $r_0$ to 0
    Let $z$ be the integer whose binary representation is $S$
    For $i$ from 1 to $s$ do
Let \( s_i \) be the \( g \)-bit binary representation of the integer \((z + i) \mod 2^g\).
Compute \( R_i = H(s_i) \)

\}

Let \( R = R_0 \| R_1 \| \cdots \| R_s \)
Let \( r \) be the integer whose binary representation is \( R \)
If \((r \cdot b^2 = a^3 \mod p)\) then
   Return “Accept”
Else
   Return “Reject”
\}

### 3.5.3 Computing the Number of Points on an Elliptic Curve

Determining the number of points on an elliptic curve is an essential ingredient of domain parameter generation. While no straightforward formula is known for calculating the group order for the EC group, there are several methods and/or algorithms available, such as Schoof’s algorithm and the BSGS method. In this section, BSGS method is used to compute the number of points of a given elliptic curve.

Let \( E \) be an elliptic curve defined over the finite field, \( F_p \). Then, the number of points in \( E(F_p) \), denoted \#\( E(F_p) \), is called the order of \( E \) over \( F_p \). Then, a bound for \#\( E(F_p) \) given by Hasse’s Theorem.

**Theorem 3.2: Hasse’s Theorem** The order of \( E(F_p) \) satisfies the following inequality

\[
|p + 1 - \#E(F_p)| \leq 2\sqrt{p}
\] (3.11)

Definition 3.3: Let $P$ be a point on $E$, then the order of $P$ is the smallest positive integer, $k$, such that $kP = O$, where $O$ is the group identity. The order of $P$ is denoted by $ord(P)$.

Lagrange’s theorem states that the order of a point $P$ in $E(F_p)$ divides the group order $#E(F_p)$, so when the order of $P$ is greater than $(p + 1 - 2\sqrt[4]{p})$, then it is said that $#E(F_p)$ is an integer multiple of that order, which lies in Hasse’s interval,

$$ (p + 1) - 2\sqrt[4]{p} \leq #E(F_p) \leq (p + 1) + 2\sqrt[4]{p} \quad (3.12) $$

If there is more than one integer multiple of the order of $P$ that lies in Hasse’s interval specified in equation (3.12), then the orders of other points, say $Q$ and $R$ are to be found. Further, the lowest common multiple of these orders is found to lie in Hasse’s interval. With this result, the order of a point $P$ is determined, and to do that, the algorithm ‘ComputeGroupOrderBSGS’ is used.

Algorithm ComputeGroupOrderBSGS( )

// Input : Elliptic Curve $E : y^2 = x^3 + ax + b$ over $F_p$
// Output : Order of Elliptic Curve $#E(F_p)$
{
    step1: Compute $Q = (p + 1)P$
    Calculate $m = \lceil p^{1/4} \rceil$
    Create the baby step set: \{P, 2P, 3P, \ldots , mP\}
    Calculate $R = 2mP$
Create the giant step set: \( \{Q - R, Q - 2R, Q - 3R, \ldots \} \) until a match is found between the baby step set and the giant step set.

i.e. \( Q - jR = Q - 2jmP = iP \), for some \( j \) and \( i \).

Conclude that \( (p + 1 - 2jm - i)P = kP = O \)

Step7: Factorise \( k = k_1k_2 \cdots k_r \)

Step8: For \( q \) from 1 to \( r \) do

\[
\begin{align*}
\text{Compute } k/k_qP \\
\text{If } k/k_qP \neq O \text{ for all } q, \text{ then } k = \text{ord}(P) \\
\text{If } k/k_qP = O \text{ for some } q, \text{ then repeat step7 with } k/k_q
\end{align*}
\]

Repeat step1 to step8 with randomly chosen points in \( E(F_p) \) until the lowest common multiple of orders divides only one integer \( n \), which lies in Hasse’s interval. Then \( n = \#E(F_p) \)

Example 3.2: Define an elliptic curve \( E: y^2 = x^3 + 4x + 10 \) over \( F_{19} \). Compute \( m \) to be \( \lceil 19^{1/4} \rceil = 3 \) and take the point \( P = (2, 8) \) on the curve. In \( F_{19} \), Hasse’s interval is \( 11.28 \leq \#E(F_{19}) \leq 28.72 \).

Next, compute \( Q = (19 + 1)P = 20P = (16, 3) \) and create the baby step set \( \{P, 2P, 3P\} = \{(2, 8), (16, 16), (18, 10)\} \). Then, calculate \( R = 2mP = 6P = (13, 6) \) and create the giant step set \( \{Q - R\} = \{(18, 10)\} \). The generating elements in giant step are stopped because there is a match between baby step and giant step.

\[
\begin{align*}
Q - R &= 3P \\
20P - 6P - 3P &= O \\
11P &= O
\end{align*}
\]
Since 11 is prime number, the order of $P$ is 11. Also, it is observed that there is only one integer multiple of $\text{ord}(P)$ that lies in the equation (3.11) which is 22. Hence, $\#E(F_{19}) = 22$.

The BSGS method is a simple method to compute the number of points in $E(F_p)$. One of the advantages of this method is that the algorithm requires $O(\sqrt{q})$ operations, where $q$ is the number of points. Thus, the group order $\#E(F_p)$ can be determined very quickly. However, when dealing with elliptic curves used in commercial cryptosystems, the preferred algorithm is the Schoof-Elkies-Atkins or Schoof’s algorithm (Schoof 1985).

Schoof presented the first polynomial time algorithm for computing $\#E(F_p)$ for an arbitrary elliptic curve $E$. The algorithm computes $\#E(F_p) \mod l$ for small prime numbers $l$, and then determines $\#E(F_p)$ using the CRT. The Schoof’s algorithm, which is the best algorithm known for counting the points on arbitrary ECs over prime fields, takes a few minutes for values of $p$ of practical interest. Since it can very quickly determine the number of points modulo small prime’s $l$, it can be used in an early abort strategy to quickly eliminate candidate curves whose orders are divisible by a small prime number. According to Washington (2003), School’s algorithm has been known to compute elliptic curve groups with order over 200 digits.

### 3.6 ELLIPTIC CURVE CRYPTOGRAPHY

The ElGamal cryptosystem can be based on any cyclic group and the same system is applied on the elliptic curve group, $E(F_p)$ (Koblitz 1987). As in Section 3.3, addition is not a straight forward process. Moreover, in Section 3.5.3, determining the group order of $E(F_p)$ requires complex algorithms. This means that with ECC, smaller keys are sufficient to provide
more security in message or data transmission. For example, the sizes of
public and private keys of the ElGamal cryptosystem are 3072-bit and 256-bit
respectively, while a 163-bit key size of an elliptic curve cryptosystem
provides the same security level as the ElGamal keys (Lauter 2004). It is for
these reasons that the NSA recommends EC cryptographic algorithms.

There are several elliptic curve cryptosystems such as the analog of
the ElGamal encryption, the analog of the RSA encryption and the Weil
Pairing encryption. This thesis emphasises on the elliptic curve ElGamal
encryption procedure.

3.6.1 Elliptic Curve Key Pairs

An elliptic curve key pair is associated with a particular set of
domain parameters $D = (q, FR, S, a, b, G, n, h)$. The public key is a randomly
selected point $Q$ in the group generated by $G$. The corresponding private key is
$d = \log_G Q$. The entity $A$ generating the key pair must have the assurance that
the domain parameters are valid. The association between domain parameters
and a public key must be verifiable by all entities who may subsequently use
$A$’s public key. In practice, this association can be achieved by cryptographic
means (e.g., a certification authority generates a certificate attesting to this
association) or by context (e.g., all entities use the same domain parameters).

Algorithm generateKeyPair( )
// Input : Domain parameters $D$
// Output : Public key $Q$, private key $d$
{
    Select $d \in R [1, n - 1]$
    Compute $Q = dG$
The problem of computing a private key $d$ from the public key $Q$ is precisely the ECDLP. Hence, it is crucial that the domain parameters $D$ be selected so that the ECDLP is intractable. Furthermore, it is important that the numbers $d$ generated be random in the sense that the probability of any particular value being selected must be sufficiently small to preclude an adversary from gaining advantage through optimizing a search strategy based on such probability.

The purpose of public key validation is to verify that a public key possesses certain arithmetic properties. Successful execution demonstrates that an associated private key logically exists, although it does not demonstrate that someone has actually computed the private key nor the claimed owner actually possesses it. Public key validation is especially important in DH based key establishment protocols where an entity $A$ derives a shared secret $k$ by combining the private key with a public key received from another entity $B$, and subsequently uses $k$ in some symmetric key protocol (e.g., encryption or message authentication). A dishonest $B$ might select an invalid public key in such a way that the use of $k$ reveals information about $A$’s private key.

**Algorithm validatePublicKey()**

// **Input** : Domain parameters $D$, public key $Q$
// **Output** : Acceptance or rejection of the validity of $Q$

```
{  
    Check that $Q \neq \infty$
    Verify that $x_Q$ and $y_Q \in F_p$
    Check that $Q$ satisfies the EC equation defined by $a$ and $b$
}
Verify that \( nQ = \infty \)

If (Verification fails) then

Return “Invalid”

Else

Return “Valid”

} 

There may be much faster methods for verifying that \( nQ = \infty \) than performing an expensive point multiplication \( nQ \). For example, if \( h = 1 \) which is usually the case for ECs over prime fields that are used in practice, then the checks for first three steps of algorithm ‘validatePublicKey’ imply that \( nQ = \infty \). In some protocols the check that \( nQ = \infty \) may be omitted and either embedded in the protocol computations or replaced by the check that \( hQ \neq \infty \). The latter check guarantees that \( Q \) is not in a small subgroup of \( E(F_p) \) of order dividing \( h \).

### 3.6.2 Elliptic Curve Encryption Scheme

For an elliptic curve ElGamal encryption, all computations are done in the finite field \( F_p \). The encryption and decryption procedures for the elliptic curve analogue on the basic ElGamal encryption scheme are presented as algorithms ‘encryptECElGamal’ and ‘decryptECElGamal’ respectively. A plaintext \( m \) is first represented as a point \( P_m \), and then encrypted by adding it to \( kQ \), where \( k \) is a randomly selected integer, and \( Q \) is the intended recipient’s public key. The sender transmits the points \( C_1 = kG \) and \( C_2 = P_m + kQ \) to the recipient who uses the private key \( d \) to compute \( dC_1 = d(kG) = k(dG) = kQ \), and thereafter recovers \( P_m = C_2 - kQ \). An eavesdropper who wishes to recover \( P_m \) needs to compute \( kQ \). This task of computing \( kQ \) from the domain parameters, \( Q \), and \( C_1 = kG \), is the elliptic curve analogue of the DH problem.
Algorithm encryptECElGamal( )

// Input : EC domain parameters \((p, E, G, n)\), public key \(Q\), plaintext \(m\)
// Output : Cipher text \((C_1, C_2)\)
{
    Represent the message \(m\) as a point \(P_m\) in \(E(F_p)\)
    Select \(k \in R_{[1, n-1]}\)
    Compute \(C_1 = kG\)
    Compute \(C_2 = P_m + kQ\)
    Return \((C_1, C_2)\)
}

Algorithm decryptECElGamal( )

// Input : EC Domain parameters \((p, E, G, n)\), private key \(d\),
        cipher text \((C_1, C_2)\)
// Output : Plaintext \(m\)
{
    Compute \(P_m = C_2 - dC_1\)
    Extract \(m\) from \(P_m\)
    Return \((m)\)
}

As in the finite field case, the security of this cryptosystem lies in the fact that if only \(G\) and \(Q\) are known to the adversary, it is difficult to determine the number of times \(G\) has been added to itself to get \(Q\). This property is due to the random additive structure of points. Koblitz (1987) mentioned that the techniques developed to solve the DLP for finite fields often fail to work for the ECDLP. This fact enables this elliptic curve cryptosystem to remain secure while keeping the size of the field small. There exist several methods but without an efficient algorithm to attack the system,
the difficulty in solving the ECDLP remains the key advantage of using ECs in cryptography.

3.6.3 Elliptic Curve Diffie Hellman Scheme

Elliptic Curve Diffie Hellman (ECDH) is a key agreement scheme that allows two entities to establish a shared secret key that can be used for private key algorithms. Both entities exchange some public information to each other. Using this public information and their own private information these entities calculate the shared secret.

For generating a shared secret between two entities $A$ and $B$ using ECDH, both have to agree upon elliptic curve domain parameters. Both entities have a key pair consisting of a private key (a randomly selected integer less than $n$, where $n$ is the order of the curve, an elliptic curve domain parameters) and a public key ($G$ is the generator point, an elliptic curve domain parameter). Let $(n_A, P_A)$ be the private key - public key pair of user $A$ and $(n_B, P_B)$ be the private key - public key pair of user $B$. Since the shared secret key $k = n_A P_B = n_A n_B G = n_B n_A G = n_B P_A = k$. The algorithm ‘ComputeECDHSecretKey’ is used to compute the shared secret key between two users $A$ and $B$.

Algorithm ComputeECDHSecretKey( )

// Input : EC domain parameters ($p, E, G, n$)

// Output : Secret key $k$

{
    User $A$ select $n_A \in R [1, n - 1]$
    User $A$ compute $P_A = n_A G$
    User $B$ select $n_B \in R [1, n - 1]$
}
User B compute $P_B = n_B G$
User A calculate $k = n_A P_B$
User B calculate $k = n_B P_A$
Return $k$

Since it is practically impossible to find the private key $n_A$ or $n_B$ from the public key $P_A$ or $P_B$, it is not possible to obtain the shared secret key $k$ for a third party.

3.7 RESULTS AND DISCUSSION

For demonstration purposes, elliptic curve is represented by $y^2 = x^3 + x + 1$ defined over $E(F_7)$, where $a = 1$, $b = 1$ and $p = 7$. The coefficients $a$ and $b$ are chosen based on the condition that $4a^3 + 27b^2 \mod p = 31 \mod 7 = 3 \neq 0$, so $E(F_7)$ is indeed an elliptic curve. The generated points on the EC can be found and they are \{O, (0, 1), (0, 6), (2, 2), (2, 5)\}.

Then consider the elliptic curve $E: y^2 = x^3 + 4x + 20$ defined over $F_{29}$ with the constants $a = 4$ and $b = 20$ which have been checked to satisfy that $E$ is indeed an elliptic curve. The 37 points in $E(F_{29})$ are the following:\{ O, (0, 7), (0, 22), (1, 5), (1, 24), (2, 6), (2, 23), (3, 1), (3, 28), (4, 10), (4, 19), (5, 7), (5, 22), (6, 12), (6, 17), (8, 10), (8, 19), (10, 4), (10, 25), (13, 6), (13, 23), (14, 6), (14, 23), (15, 2), (15, 27), (16, 2), (16, 27), (17, 10), (17, 19), (19, 13), (19, 16), (20, 3), (20, 26), (24, 7), (24, 22), (27, 2), (27, 27) \}. The point (1, 5) in $E(F_{29})$ satisfies the equation (3.2) since:
\[
y^2 \mod p = x^3 + 4x + 20 \mod p \\
25 \mod 29 = 1 + 4 + 20 \mod 29 \\
25 = 25\]
Similarly, other generated EC points also satisfy the equation. These points are graphed in the following Figure 3.3.

![Figure 3.3 Elliptic curve point representation](image)

Note that there are two points for every \( x \) value. Over the field of \( F_{29} \), the negative components in the \( y \)-values are taken modulo 29, resulting in a positive number as a difference from 29. Here \(-P = (x_1, (-y_1 \mod 29))\). Based on the field size \( p \), the number of points on the EC can be varied.

Point addition and point doubling are the basic EC operations. Given EC, \( E: y^2 = x^3 + x + 1 \) over \( F_{13} \). The group \( E(F_{13}) \) has 18 elements.

\[
\{ \text{O}, (0, 1), (0, 12), (1, 4), (1, 9), (4, 2), (4, 11), (5, 1), (5, 12), (7, 0), (8, 1), (8, 12), (10, 6), (10, 7), (11, 2), (11, 11), (12, 5), (12, 8) \}
\]
Consider two points \( P = (12, 8) \) and \( Q = (1, 9) \) on \( E(F_{13}) \). The addition of two points \( P + Q = (x_3, y_3) \) where \( P \neq \pm Q \) is computed as follows:

First calculate \( \lambda \) to be:
\[
\lambda = (9 - 8) / (1 - 12) = 1 / (-11) = 1 / 2 \equiv 7 \pmod{13}
\]

Then, using the formulae in equation (3.4) to calculate the coordinates as
\[
x_3 = 72 - (12 + 1) \equiv 10 \pmod{13}
\]
\[
y_3 = 7(10) + 2 \equiv 7 \pmod{13}
\]

So, \( (12, 8) + (1, 9) = (10, 7) \) which also lies on the elliptic curve \( E(F_{13}) \).

Consider a point \( P = (11, 2) \) on \( E(F_{13}) \). To add a point to itself that is double a point \( 2P = (x_3, y_3) \). First found \( \lambda \) to be:
\[
\lambda = 3(11^2) + 1 / 2 \times 2 \equiv 0 \pmod{13}
\]

Then using the formulae in equation (3.5) to calculate the coordinates as
\[
x_3 = 0 - (2 \times 11) \equiv 4 \pmod{13}
\]
\[
y_3 = 0 + 2 \equiv 2 \pmod{13}
\]

So, \( 2 \times (11, 2) = (4, 2) \) which lies on \( E(F_{13}) \).

Cryptographic schemes based on ECC rely on scalar multiplication of EC points. Let \( P \) is a point on an EC, and one needs to compute \( kP \), where \( k \) is a positive integer. This scalar multiplication can be computed efficiently by a series of doubling and addition of \( P \). For example, given \( k = 13 \), entails the following sequence of operations, by which the efficiency of the scalar multiplication of the points is improved.

<table>
<thead>
<tr>
<th>( P )</th>
<th>( 2P )</th>
<th>( 3P )</th>
<th>( 6P )</th>
<th>( 12P )</th>
<th>( 13P )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Doubling</td>
<td>Addition</td>
<td>Doubling</td>
<td>Doubling</td>
<td>Addition</td>
<td></td>
</tr>
</tbody>
</table>
For example, consider the point $P = (12, 8)$ on the elliptic curve $E(F_{13})$. Based on the algorithm ‘computeScalarMul’ the value of $13P$ is computed as $(1, 4)$ which also lies on the elliptic curve $E(F_{13})$.

The following example demonstrates the encryption and decryption processes using EC. Consider the EC $y^2 = x^3 - 5x + 25 \mod 487$. Here, $a = -5$, $b = 25$ and $p = 487$ are the parameters of EC. Using equation (3.2) the points are generated. The base point $G$ of an EC is selected as $(0, 5)$. Assume that the user $A$ wants to send the message 48 to user $B$. First choose the random point on EC as $(1, 316)$. The message is encoded as point on EC as $(12, 233)$.

User $B$ chooses the private key $n_B$ as 277. The public key $P_B$ is computed according to the algorithm ‘generateKeyPair’ as $(260, 48)$. According to algorithm ‘encryptECElGamal’, user $A$ chooses $k$ as 225 and compute $C_1$ as $(0, 5)$. Then compute $C_2$ as $(12, 233) + (260, 48) = (384, 288)$. Therefore, the cipher text $C = (C_1, C_2) = ((0, 5), (384, 288))$. According to algorithm ‘decryptECElGamal’, compute encoded message EC point as $(384, 288) - 277(0, 5) = (12, 233)$. Then extract the message from $(12, 233)$ using discrete logarithm concept as 48.

The following example demonstrates how two users generate a secret key using ECDH. Consider the EC $y^2 = x^3 - 5x + 25 \mod 487$. Here, $a = -5$, $b = 25$ and $p = 487$ are the parameters of EC. User $A$ select the private key $n_A$ as 719 and its public key is computed according to the algorithm ‘generateKeyPair’ as $P_A = n_A G = 719(0, 5) = (213, 351)$. Similarly, user $B$ choose the private key $n_B$ as 967 and its public key is calculated as $P_B = n_B G = 967(0, 5) = (114, 364)$. Based on the algorithm ‘ComputeECDHSecretKey’, the shared secret key $k = n_A P_B = n_B P_A = 719(114, 364) = 967(213, 351) = (195, 469)$. 
3.8 SUMMARY

It is seen in this chapter that the points on an EC over an arbitrary field form a group. This was mainly because the group law for addition can be expressed formally by algebraic equations which can be applied over any field. This group of points is very crucial in the implementation of the elliptic curve cryptosystems. Since, by the algebraic formulae, the group operations eventually mount to computations in the field where the EC is defined, one has to choose a field with an efficiently implementable arithmetic. Basically, this requirement narrows down to the finite fields. In this chapter, EC over finite prime field is discussed in particular and the results are used for further enhancement of data security.