CHAPTER 1

INTRODUCTION

This chapter gives the introduction to Cryptography and Digital Signature. Followed by, it provides an overview for Elliptic Curve Cryptography and Elliptic Curve Digital Signature Algorithm and also lists out the different applications of these algorithms.

1.1 CRYPTOLOGY

Cryptology is a relationship between the Cryptography and Cryptanalysis. The cryptography is a methodology to play with encryption and decryption techniques, whereas, the cryptanalysis is the process to analyze the strength of cryptography against on the relationship among the original information, hidden information and key values (Chan KS et al 2004). The following Venn diagram in Figure 1.1 shows the relationship among the cryptology, cryptography and cryptanalysis.

Figure 1.1 Venn diagram to understand the relationship among cryptology, cryptography and cryptanalysis
1.1.1 Cryptography

Cryptography is a combination of encryption and decryption techniques. An encryption means to generate the hidden message from the original message, whereas, the decryption is the reverse process of encryption. Both are performed with the help of key values. Here, the original information is a plain text and the hidden information is a cipher text shown in Figure 1.2. The common mathematical notations of cryptography are as follows:

\[ P \rightarrow \text{Plain Text} \]
\[ C \rightarrow \text{Cipher Text} \]
\[ E \rightarrow \text{Encryption process} \]
\[ D \rightarrow \text{Decryption process} \]
\[ k \rightarrow \text{Key value} \]
\[ S \rightarrow \text{Sender} \]
\[ R \rightarrow \text{Receiver} \]
\[ C = E_k(P) \]
\[ P = D_k(C) \]

![Figure 1.2 Block diagram of encryption and decryption processes on the text in network](image-url)
The main objectives of cryptography are to provide the following security services (Jena D et al 2007):

**Secrecy**: It is used to confirm the confidentiality of information

**Integrity**: It is used to check the sequence of information

**Authentication**: It is used to check the privacy of information

**Notarization**: It is used to confirm the message delivery

Most of the cryptographies provide the secrecy, authentication and notarization services. Normally, the integrity is included by the way of processing text through encryption and decryption. For example, one of the methods is the cipher / plain text feedback block principle. Another way to provide integrity is Digital Signature Algorithm (DSA). Besides these security services avoid the different types of attacks on network as shown in Figure 1.3 (Ekambaram Kesavulu Reddy 2011).

**Types of Security attacks**

![Diagram of types of security attacks](image)

**Figure 1.3 Different types of security attacks on the network**
These attacks are generally classified into passive attacks and active attacks (William Stallings 2009). A passive attack does not affect the flow of information, but it monitors the flow of information on network. So there is no privacy for information. Some of the examples are the traffic analysis and message type analysis. The second type of attack is an active attack which affects or alters the flow of information. It will make harmful to another side for predicting the information. So it is more dangerous than passive attack. For example, the masquerade, replay, information modification and denial of service are active attacks.

1.1.2 Types of Cryptography

The cryptographies are categorized into different types, based on the functionalities, processing methods and the number of keys. The first category is functionality type which defines the way of mapping the plain text and the cipher. It is implemented through the substitution and transposition operations. Here, the substitution replaces a bit, byte or word with another bit, byte or word and the transposition changes the position of a bit, byte or word with another bit, byte or word position. The second category is the processing methods which handles a plain text or a cipher text in terms of a bit, byte or word. Based on this, it is classified into two different ways. One is a block cipher and the other is stream cipher. The block cipher concurrently performs encryption or decryption whereas the stream cipher performs in sequential manner. The last category depends upon the number of keys. Generally, it is classified into symmetric key cryptography and asymmetric key cryptography. A symmetric key performs the encryption or decryption based on a common secret key, but the asymmetric key cryptography is defined with a secret key for the same operations. Here, the secret key is computed based on the known public key and secret private key (Hung-Min Sun et al 2004).
Today, a secure environment for information exchange is provided, based on the symmetric key or asymmetric key cryptography for different levels of securities such as a Level-1 (commercial), Level-2 (confidential), Level-3 (secret) and Level-4 (top secret). In the symmetric key cryptography, the secret key may be generated by a sender, a receiver or a trusted party in the communication. If the number of persons in the communication is ‘n’, it will require n(n-1)/2 number of secret keys for this type of cryptography. The other names of this cryptography are: the private key cryptography, secret key cryptography or conventional cryptography, and some of the examples are DES (Data Encryption Standard) and AES (Advanced Encryption Standard) (Burke J et al 2000). It consists of the key generation with distribution, encryption and decryption procedures. These procedures are diagrammatically explained in Figure 1.4 and its corresponding mathematical notations are described as follows:

Figure 1.4 Block diagram of symmetric key cryptography for encryption and decryption operations

A symmetric encryption is a process to convert the plain text (P) into cipher text (C) by using a secret key (k) which is generated through a key generation module denoted as follows:

\[ C = E_k(P) \]
The reverse process of encryption is decryption which converts the cipher text (C) into plain text (P) by using the same secret key (k). In this case, both the processes share a common secret key.

\[ P = D_k(C) = D_k(E_k(P)) = P \]

Both the operations need a secret key which is generated separately through the key generation module, and this key is distributed to the corresponding communication places. It is also called Private Key Management.

The second type of cryptography based on the number of keys is an asymmetric key cryptography. It is also known as a public key cryptography. It uses the combination of public key (PU) and private key (PR) to generate a secret key. It also consists of the key generation and distribution, encryption and decryption (Zhenfu Cao et al 2009). In this case, each communication party maintains the number of public keys (n) and its own private key. So the total number of keys on each place is ‘n+1’ where, n is the number of public keys with a private key (Delgosha F et al 2006). The popular examples of this type are RSA (Rivast-Shamir-Adleman) and ECC (Elliptic Curve Cryptography) and the different mathematical notations of this cryptography are as follows:

- PU → Public Key
- PR → Private Key
- PU_s → Sender’s Public key
- PR_s → Sender’s Private key
- PU_r → Receiver’s Public key
- PR_r → Receiver’s Private key
Here, the plain text is converted into cipher text, based on the private key of the sender and the public key of the receiver. These keys are generated and distributed through the public key management.

\[ C = E_k(P) \text{ where, } k = PU_R + PR_S \]

The decryption converts the cipher text into plain text, based on the private key of the receiver and the public key of the sender.

\[ P = D_k(c) \text{ where, } k = PU_S + PR_R \]

In the public key management, the key generation and key distribution modules generate a pair of keys which consist of a private key and a public key. Besides, the block diagram of asymmetric key cryptography is shown in Figure 1.5.

Sender → \{PU_S, PR_S\} keys  
Receiver → \{PU_R, PR_R\} keys  
(secret key on sender side) \( k = PU_R + PR_S \)

\[ PU_S + PR_R = k \text{ (secret key on receiver side) } \]

---

**Figure 1.5** Block diagram of asymmetric key cryptography for encryption and decryption operations
Both the symmetric and asymmetric cryptographies have a set of merits and demerits. Some of the asymmetric key cryptography merits become the demerits of symmetric key cryptography, and some of the symmetric key cryptography merits become the demerits of symmetric key cryptography (Rodrigo Roman 2007). The differences between these two cryptographies are shown in Table 1.1.

**Table 1.1 Differences between symmetric key and asymmetric key cryptographies**

<table>
<thead>
<tr>
<th>Symmetric Key Cryptography</th>
<th>Asymmetric Key Cryptography</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single key for encryption</td>
<td>Two keys for Encryption</td>
</tr>
<tr>
<td>Shared by sender and receiver</td>
<td>One key is shared (public key) and another key is personal one.</td>
</tr>
<tr>
<td>Encryption and decryption are the same algorithm</td>
<td>Encryption and decryption are different algorithms</td>
</tr>
<tr>
<td>Total keys on the network: (\frac{n(n-1)}{2})</td>
<td>Total keys: (n + 1)</td>
</tr>
<tr>
<td>Maintain a data base for secret key</td>
<td>Maintain two databases. One is for public key database and another is for private key database</td>
</tr>
<tr>
<td>Needs less space to store keys</td>
<td>Needs more space to store keys</td>
</tr>
<tr>
<td>Less time needed to encryption</td>
<td>More time needed to encryption</td>
</tr>
<tr>
<td>Less security compared to asymmetric key cryptography</td>
<td>More security compared to symmetric key cryptography</td>
</tr>
<tr>
<td>High possibility for key leakage</td>
<td>Less possibility for key leakage</td>
</tr>
<tr>
<td>Suitable for short and long messages</td>
<td>More suitable for short messages</td>
</tr>
<tr>
<td>Symbols are permuted and substituted</td>
<td>Numbers are manipulated</td>
</tr>
</tbody>
</table>
The main objective of asymmetric key cryptography is to provide better security based on the key, at the same time, to reduce the number of keys remembered on each place. The second objective is to increase the security level of algorithm through the key pairs. But the main drawback of this cryptography is the time and the space complexities of algorithm (Bertoni G 2006).

### 1.1.3 Cryptanalysis

Cryptography is an art to create secret codes, whereas, a cryptanalysis is a group of methods to break secrecy for finding the original information and key values from the captured information. This cryptanalysis is mainly used to analyze the strength of cryptography mathematically, based on the different constrains. The different types of attacks are also analyzed through the cryptanalysis (Forouzan BA 2008). These attacks are shown in Figure 1.6 and the definition of each type is as follows:

![Figure 1.6 Different types of security attacks based on cryptanalysis](image)

Figure 1.6 Different types of security attacks based on cryptanalysis
Cipher text only attack:

The hacker or cryptanalyst captures the cipher text from the network and tries to find out the plain text, key or both. In this case, the hacker only knows an encryption algorithm and the cipher text. They use different methods such as brute-force, statistical and pattern attacks to find the original information.

Known plain text attack:

The hacker or cryptanalyst uses the plain and cipher text pairs to find out the key. In this case, they know an encryption algorithm, the cipher text and some part of plain text with the secret key.

Chosen cipher text attack:

It is similar to the plain text attack. But the hacker and cryptanalyst have chosen a pair of the cipher and plain text, and find out the key and the cipher text.

Chosen plain text attack:

It is similar to the plain text attack. But the hacker and cryptanalyst have considered a pair of plain and cipher text to find out the key and the plain text.

Chosen text attack:

It is a combination of the chosen plain and cipher text. The hacker or cryptanalyst has chosen the plaintext with cipher text to find out the key, plain text or cipher text on the communications.
The strength of an encryption, decryption or both are tested against these attacks (Zuhua Shao 2009). The verification and validation of testing are conducted computationally and conditionally (William Stalling 2007).

1.2 DIGITAL SIGNATURE

Digital Signature is a security mechanism controlled by the trusted third party known as the Certificate Authority (CA). This mechanism generates the digital certificate and it is appended with the original information. This message will be verified on another side to confirm the integrity of message. The FIPS 186-3 (Federal Information Processing Standard) Digital Signature Algorithm Validation System Document has specified the different procedures of configurations for validating the implementations of the Digital Signature Algorithm (DSA). It also specifies the basic design of DSA algorithm (Jianhong Zhang et al 2008).

Figure 1.7 Block diagram of digital signature algorithm to perform message authentication in secure environment
It consists of the domain parameter generation with verification, key pair generation and signature generation with verification procedures. These components provide the different security services for message authentication. The combination of message authentication and cryptography are used to implement the DSA algorithm as shown in Figure 1.7.

1.2.1 Message Authentication

The message authentication is a procedure to identify the corresponding communication parties on the network without any loss of message. It provides data integrity, authentication and non-repudiation services for the message during transaction based on the verification code. But it does not provide data confidentiality for this message. The verification code is derived from Message Detection Code (Westoff D et al 2005).

The Message Detection Code (MDC) is a procedure to generate secret code, which is known as message digest. It is transmitted through a trusted third party on the other side with the original information. In the receiving side, it is separated as the original information and the secret code. Once again, the secret code is generated based on the received information and it is verified with already generated received secret code value. If both values are equal, the message will be accepted. It means that there are no changes during data transfer and confirms the data integrity of message. But this procedure does not support for data origin authentication and peer data authentication. It is actually derived from some of the error detection techniques such as Cyclic Redundancy Check (CRC) and check sum (Alexander W Dent et al 2010).

The Message Authentication Code is implemented with a private key of sender and public key of receiver to produce the secret code. This code is attached with message and it is sent to the receiver side. The receiver
separates them into the message and secret code and regenerates the secret
code based on his private key, sender public key and this message. Then it is
verified with separated secret code. If both values are equal, it will give the
right to access the message. The Message Digest algorithms and Hash
functions are examples of this type.

1.2.2 Secure Hash Algorithm

The hash function is a mathematical transformation from a long
arbitrary message (m) into a fixed short length code h(m). It is denoted as
follows:

\[
m \rightarrow \text{the lengthy message}
\]

\[
h(m) \rightarrow \text{the fixed short hash code that is generated by the hash function}
\]

The hash is mainly based on the following three properties of
mathematics (Kaps J 2005):

1. Easy to compute \( y = h(x) \), but difficult to compute \( h(y) = x \)
2. Compute \( y = h(x) \) and \( x = y \), then it will be \( h(x) = h(y) \)
3. The two values of \( x \) and \( y \) are not equal, and it will be \( h(x) = h(y) \)

The Secure Hash Algorithm (SHA) is developed by NIST (National
Institute of Standards and Technology) and NSA (National Security System).
Later it is combined with cryptography to define the Digital Signature
Standard (DSS). The specification of this methodology is known as a FIPS
Publication 180: Secure Hash Standard (SHS) of NIST in 1993. The main
objectives of this standard are as follows:
1. Specify the secure hash algorithm to be required for the use of the Digital Signature Standard operations such as the digital signature generation, verification and key generation.

2. Specify the secure hash algorithm to be used, whenever a secure hash algorithm is required based on FIPS documentation.

3. Encourage the adoption and use of the specified secure hash algorithm for private and commercial organizations.

The SHA-1 may be implemented in the form of software, firmware or hardware (Henk CA Van Tilborg et al 2005). This algorithm is explained by the following steps.

1. Initialize five variables: hash₁, hash₂, hash₃, hash₄ and hash₅.

2. Appending the starting bit of message, padding bits and specifying the length of message should be pre-processed.

3. Process the message in blocks which has 512-bits length.
   3.1. Each block has sixteen 32-bit words w[i], 1 ≤ i ≤ 16
   3.2. Extend the sixteen 32-bit words into eighty 32-bit words:
   3.3. Initialize hash value of these words:
   3.4. Repeat 80 times the following:
       3.4.1 Find out intermediate values (i₁, i₂, i₃, i₄, i₅, i₆) for word processing.
       3.4.2 Each time, add those word hash to result.
           3.4.2.1. hash₁= sum of hash₁ and i₁
           3.4.2.2. hash₂= sum of hash₂ and i₂
           3.4.2.3. hash₃= sum of hash₃ and i₃
           3.4.2.4. hash₄= sum of hash₄ and i₄
           3.4.2.5. hash₅= sum of hash₅ and i₅
4. \[ \text{digest} = \text{the combined hash values} \]
\[ = (\text{hash}_1)_\text{hash}_2)_\text{hash}_3)_\text{hash}_4)_\text{hash}_5) \]
\[ = i_6 \]

The SHA-1 is mainly used in DSA for electronic Mail, electronic Funds Transfer, Software Distribution, Data Storage, SSL (Secure Sockets Level), PGP (Pretty Good Privacy), XML (Extensible Markup Language) Signatures and other applications which requires the data integrity assurance or data origin authentication (Vineeta Khemchandani et al 2010). This algorithm is diagrammatically shown in the following Figure 1.8.

---

**Figure 1.8 Different functions of SHA-1 algorithm to generate hash code**

---

1.2.3 Digital Signature Operations

A Digital Signature (DS) is a message authentication procedure to sign the document by the signer. This sign is verified on another place by the verifier electronically. The sign is created by using signer private key and it is verified by using signer public key. The procedure of DS is implemented by
using asymmetric key cryptography (Yang Shi et al 2004). It is mainly classified into two types. They are: one way authentication of message delivery (connectionless service) and two way authentications (connection oriented service) of interactive communication. The different mathematics notations of this concept are as follows:

\[ \begin{align*} 
M & \rightarrow \text{Message} \\
S & \rightarrow \text{Signer} \\
V & \rightarrow \text{Verifier} \\
S_1, S_2 & \rightarrow \text{Signature value} \\
v & \rightarrow \text{verification code} \\
MD & \rightarrow \text{Message Digest} \\
h(m) & \rightarrow \text{Hash function} \\
r & \rightarrow \text{random secret} \\
d & \rightarrow \text{private key} \\
(e_1, e_2, p, q) & \rightarrow \text{public key} \\
p, q & \rightarrow \text{prime numbers} \\
e_0 & \rightarrow \text{element in } Z_p \\
e_1 & \rightarrow e_0^{(p-1)/q} \mod p \\
e_2 & \rightarrow e_1d \\
\text{PU} & \leftarrow e_1, e_2, p, q \\
\text{PR} & \leftarrow d 
\end{align*} \]

The main objective of DS is to provide data integrity, authentication and notarization along with confidentiality for information. If the message and sign are encrypted, then it is known as Digital Signature Algorithm (DSA). The examples of this type are: RSA based DSA and ECC.
based DSA. Each type consists of the key generation, digital signing and verification modules (Asokan N et al 2000). The key generation is a procedure to generate a pair of keys for digital signing and verifying processes. It is explained with the following steps:

1. Assume a prime $p$ which is a multiple of 64 bits
2. Assume another prime $q$ which is 160 bits and $q$ divides by $p-1$
3. Define two multiplication group $\mathbb{Z}_p^*$ and $\mathbb{Z}_q^*$ with two operations $+$ and $\times$ and $\mathbb{Z}_q^*$ is a subgroup of $\mathbb{Z}_p^*$
4. Assume $e_0$ as primitive element of $\mathbb{Z}_p$
5. Find out $e_1 = e_0^{(p-1)/q} \mod p$
6. Assume $d$ as a private key
7. Calculate $e_2 = e_1^d$
8. Assume a public key as $(e_1, e_2, p, q)$

Figure 1.9 Block diagram of DSA signature generation

Using these keys, the procedure of digital signing procedure generates two signatures on sender side, and the following steps are used to define this process shown in Figure 1.9.
1. Assume a random number \( r \) (\( 1 \leq r \leq q \)) for each time signing document.

2. Find out \( S_1 = (e_1 r \mod p) \mod q \).

3. Find out message digest \( MD = h(M) \).

4. Find out \( S_2 = (h(M) + d \cdot S_1) r^{-1} \mod q \).

5. Digital Message has \( M, S_1 \) and \( S_2 \). Then it is sent to the verifier.

On the receiver side, the procedure of digital verification is used to validate sender’s signature with generated receiver’s signature. The following steps are used to describe this operation and diagrammatically shown in Figure 1.10.

1. Check \( S_1 \) if \( 0 < S_1 < q \).
2. Check \( S_2 \) if \( 0 < S_2 < q \).
3. Find out \( MD \) using the same hash algorithm and message.
4. Find out \( v = [(e_1 h(M)/s_2 e_2 s_1)/s_2] \mod p \) \mod q \).
5. If \( MD \) and \( v \) are equal, the message is accepted, otherwise rejected.

![Figure 1.10 Block diagram for DSA signature verification](image)
These two procedures are mathematically verified and validated as following (Hung Min Sun et al 2004).

\[
V = (e_1^{(h(m)/s_2)} \cdot e_2^{(S_1/s_2)}) \mod p \mod q
\]

\[
= (e_1^{(h(m)/s_2)} \cdot e_1^{(S_1 d/S_2)}) \mod p \mod q
\]

\[
= (e_1^{(h(m)+d s_1)/s_2}) \mod p \mod q
\]

\[
= (e_1^{S_2 r/S_2}) \mod p \mod q
\]

\[
= (e_1^r) \mod p \mod q
\]

\[
= S_1 \text{(Accepted)}
\]

1.3 ELLIPTIC CURVE CRYPTOSYSTEM

The secure environment for e-Transactions is mostly created using asymmetric key cryptography. It may be short or long message applications. For example, a digital signature is an example for short message application and the secure e-Mail for long message application. Based on the mathematics, the asymmetric algorithms are also grouped into three categories as follows (Zhi-Gang Chen et al 2007):

1. Integer Factorization Problem (IFP)
2. Discrete Logarithm Problem (DLS)
3. Elliptic Curve Discrete Logarithm Problem (ECDLP)

The RSA algorithm is an example of IFP. It is the strongest algorithm which is based on the prime numbers computation. But it needs more power to manipulate the largest prime number. The example for DLS is a Diffie Hellman Key Exchange. But it is easily breakable because of its exponential term exchanges. Finally, the ECC is an example of ECDLP which provides more security, based on the mathematical background. It needs a
small key size for encryption to provide the same level of security, when it is compared with RSA. The comparison between RSA and ECC algorithms are as follows in Table 1.2 (Sakiyama K et al 2007).

### Table 1.2 Comparison between the RSA and ECC algorithm

<table>
<thead>
<tr>
<th>RSA</th>
<th>ECC</th>
</tr>
</thead>
<tbody>
<tr>
<td>The large key size provides more security, but the small key size is inefficient</td>
<td>The small key size provides greater security</td>
</tr>
<tr>
<td>It is more suitable for long messages</td>
<td>It is more suitable for long as well as short messages</td>
</tr>
<tr>
<td>Needs more power for processing a larger value of prime number</td>
<td>The processing overhead of ECC is reduced. So it consumes only less power</td>
</tr>
<tr>
<td>Needs more space to store keys</td>
<td>Needs less space</td>
</tr>
<tr>
<td>Needs more bandwidth for key transaction</td>
<td>Needs less bandwidth</td>
</tr>
</tbody>
</table>

This table concludes that the RSA is not suitable for encrypting the long messages. At the same time, the ECC algorithm supports, both long and short messages with a small key. This key provides greater security, when it is compared to RSA as shown in Table 1.3. So it is used in many applications such as e-Mail enhancement, e-Com, e-Transactions and so on (Zhu Liehuang et al 2006).
Table 1.3 Comparison among different cryptography for providing security based on key size

<table>
<thead>
<tr>
<th>RSA and Diffie-Hellman (other algorithms asymmetric Key Size) (bits)</th>
<th>Elliptic Curve Cryptography Key Size (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>160</td>
</tr>
<tr>
<td>2048</td>
<td>224</td>
</tr>
<tr>
<td>3072</td>
<td>256</td>
</tr>
<tr>
<td>7680</td>
<td>384</td>
</tr>
<tr>
<td>15360</td>
<td>521</td>
</tr>
</tbody>
</table>

1.3.1 Basics of Elliptic Curve Over Finite Field

ECC was invented by Neil Koblitz and Victor Miller in 1985. It has been studied extensively for 27 years. It is well recognized and accepted world-wide (Hasan MA 2001). It has been standardized by many organizations such as ISO (International Standard Organization), IETF (Internet Engineering Task Force), ANSI (American National Standards Institute), NIST (National Institute of Standards and Technology) and FIPS (Federal Information Processing Standard) (William Stallings 2006).

For example, The ANSI X9.62 standard defines the detailed description of ECDSA over prime and binary fields with different key sizes. It recommends ECDSA with the hash algorithm such as SHA-1. The ANSI X9.63 standard explains the key transport schemes and the key agreement based on ECC. It is another version of ANSI X9.62 with various constraints and limitations of SHA-1. It also explains Elliptic Curve domain parameters and pseudorandom number generator (Forouzan BA 2008).
The FIPS 186-2 standard defines the signature schemes released in 2000 and it is specified in ANSI X9.62. The IEEE P1363 standard is based on the traditional discrete logarithm problem and integer factorization problem. The ECDLP has a wide scope encompassing schemes, which explains the general descriptions of Elliptic Curve Signature Standard Algorithm (ECSSA) and Elliptic Curve Integrated Encryption Standard (ECIES). The IP Sec standard supports a variant of Elliptic Curve Diffie Hellman (ECDH) for key exchange. The ISO 14888-3 standard discusses the general signature mechanisms, and the draft of ISO 15946 documents are discussed with the cryptographic techniques based on ECC. These standards have a variety of signature schemes including ECDSA and a variety of key establishment schemes such as Elliptic Curve Menezes-Qu-Vanstone (ECMQV) and ECDH.

The Elliptic Curve Cryptography is basically more complex than RSA and DH (Diffie Hellman). Its performance is directly proportional to the length of the key and the complexity of mathematics. This mathematics is an algebraic structure of polynomial over finite field defined through real numbers.

1.3.1.1 Elliptic curve over real numbers

Elliptic Curve (EC) is a cubic equation. It has two variables \((x,y)\), that are used to measure the length of a curve in the circumference of an ellipse. But it is not an Ellipse equation. The general equation for an Elliptic Curve is as follows (Jithra Adikari et al 2008).

\[
y^2 + a_1xy + a_2y = x^3 + a_3x^2 + a_4x + a_5
\]  

(1.1)

where \(a_1, a_2, a_3, a_4, a_5, x \) and \(y \in \text{real numbers} \)
The above mentioned Equation (1.1) is simplified as follows for EC over real numbers (Estes Matthew et al 2006).

\[ y^2 = x^3 + a_1 x + a_2 \] (1.2)

The co-efficient values of \( a_1 \) and \( a_2 \) in Equation (1.2) are substituted in the following Equation (1.3) to find out the \( \Delta \) value (Ekambaram Kesavulu Reddy 2011).

\[ \Delta \neq 4a_1^3 + 27a_2^2 \] (1.3)

If the value of ‘\( \Delta \)’ is equal to zero, then it is called as singular equation. It does not have three distinct root values. If the value of ‘\( \Delta \)’ is not equal to zero, then it is nonsingular equation. It has three distinct root values in the form of real and complex numbers. The roots of this Equation (1.3) are real numbers, then it is called by EC over real numbers and it is denoted by \( E_p(a_1, a_2) \) where \( p \) is prime number.

Figure 1.11 Elliptic Curve equation of \( y^2 = x^3 - 25x \) over Finite Field
For example, the Equation \( y^2 = x^3 - 25x \) is assumed as EC equation, then the roots of cubic equation are -5, 0 and 5 as shown in Figure 1.11. These values are used to define the Elliptic Curve over finite field along with the polynomial equation. The basic of finite field is as follows in the next section.

### 1.3.1.2 Finite Field

The Finite Field is more important for cryptography to define the variables and coefficients over the addition and multiplication operations (Savas E 2010). So the ECC is defined through Elliptic Curve (EC) over real numbers with point operations as shown in Figure 1.12.

![Figure 1.12 Block diagram of finite field over Elliptic Curve](image)

**Figure 1.12 Block diagram of finite field over Elliptic Curve**
A Finite Field is defined with the help of a Set, Group, Ring and Field rules using Elliptic Curve points. These points are generated based on the substitution of x and y values in Equations (1.2) and (1.3) (Hankerson D 2009).

1.3.1.3 Finite field over prime numbers

The theory of Finite Field is a main part of mathematical theory in cryptology and it is used to implement EC over prime and binary fields. It is derived from the Abelain Group, Commutative Ring and Field (Forouzan BA 2008). They are defined as follows:

Definition-1: A set is a collection of elements that is denoted by Z.

Definition-2: A Group (G,*) is a set of elements over operation * with the following group axioms. Here, The operation is a addition (+).

1. \( a + b \in G \)
2. \( (a+b)+c = a+(b+c) \in G \)
3. \( a+(-a) = (-a)+a = 0 \in G \)
4. \( a+0 = 0+a = a \in G \) where a, b, c, 0, -a, -b, -c \in G

Definition-3: A Group with commutative axiom is known as Abelian Group. The commutative axiom is \( a+b=b+a \in G \)

Definition-4: A Ring \((R,+,\times)\) is an algebraic structure of G with two operations. They are known as addition and multiplication.
1. \(a \times b \in G\)

2. \((a \times b) \times c = a \times (b \times c) \in G\)

3. \((a + b) \times c = (a \times c) + (b \times c) \in G\)

**Definition-5:** A Ring with the following commutative axiom is known as a Commutative Ring.

\[a \times b = b \times a \in G\]

**Definition-6:** A Commutative Ring with the following axioms is known as integral domain.

1. \(a \times 1 = 1 \times a = a \in G\)

2. \(a \times b = 0 \in G\) than either \(a = 0\) or \(b = 0\)

**Definition-7:** A Field is defined by using an Integral Domain with the following axiom.

\[a \times a^{-1} = a^{-1} \times a = 1, a \text{ and } a^{-1} \in G\]

**Definition-8:** Finally a finite field \(GF(p)\) consists of the \(p\) field elements along with the field operations such as addition and multiplication. It is also called by Galois Field.

A Galois field \(GF(p)\) is defined with \(p\) elements. They are determined based on a prime or power of a prime numbers. It means that \(GF(p)\) is prime field of order \(p\) which contains \(p\) elements in the form of residue classes modulo \(p\) or mod \(p\) (0 to \(p\)-1) (William Stallings 2006).
1.3.1.4 Elliptic curve over prime field

The different values of point \((x,y)\) are substituted in Equations (1.2) and (1.3) to generate Elliptic Curve points for a finite field. Followed by, the operations of finite field such as point addition and point multiplication are defined on Elliptic Curve. In this case, the point addition is performed based on the following four rules (Sining Liu et al 2007).

Rule 1: \(P_1=(x_1,y_1), P_2=(0,0)\) where \(P_2\) is additive identity and \(P_1+P_2\).

\[
P_{1}+P_{2}=P_{1}
\]  

\[\text{(1.4)}\]

Figure 1.13 Point addition between point \((p)\) and it’s inverse \((-p)\)

Rule 2: \(P_1=(x_1,y_1)\) and \(P_2=\text{-}P_1=(x_1,\text{-}y_2)\) is Inverse element.

The inverse point of \(P_1\) is a negative value of ‘\(y\)’ and the same value of ‘\(x\)’.
\[ P_1 + P_2 = P_1 + (-P_1) \]
\[ = (x_1, y_1) + (x_1, -y_2) \]
\[ = (x_1, 0) \]
\[ = O \] (1.5)

It means that points are connected by using vertical line as shown in the above Figure 1.13.

Rule 3: \( P_1 = (x_1, y_1), \ P_2 = (x_1, y_1) \) and \( P_1 = P_2. \)

\[ P_3 = P_1 + P_2 \]
\[ = P_1 + P_1 \]
\[ = P_3 \] (1.6)

where \( x_3 = \frac{x_2 - x_1 - x_2}{2}, \ y_3 = \frac{x_1 - x_3 - y_1}{2} \) and \( \lambda = \frac{(3x_1^2 + a)}{2y_1} \)

Figure 1.14 Point addition using the same point P
Sometimes, this computation is known as point doubling. The slope value is calculated based on the points $P_1$ and $P_2$. In this case, the $P_1$ and $P_2$ have equal values. It is diagrammatically shown in Figure 1.14.

Rule 4: $P_1=(x_1, y_1)$, $P_2=(x_2, y_2)$ and $P_1 \neq P_2$.

$$P_3 = P_1 + P_2 = P_3$$

where $x_3 = \lambda^2 - x_1 - x_2$, $y_3 = \lambda (x_1 - x_3) - y_1$ and $\lambda = (y_2 - y_1)/(x_2 - x_1)$

This computation is called point addition. Here, the slope value is calculated by using the different points $P_1$ and $P_2$. It is shown in Figure 1.15 (Malhotra K et al 2007).

![Graph](image)

Figure 1.15 Point addition between the different points $P$ and $Q$

The second operation of EC over finite field is a point multiplication. It is calculated based on the repeated number of point addition and the point addition may be computed through the point addition or point doubling.
The number of elements in the finite field is equal to the number of points (N) in the Elliptic Curve $E_p(a,b)$. It is surrounded with the following upper and lower limits.

\[
\begin{align*}
\text{(Number of Points)} & \\
p + 1 - 2 \sqrt{p} & \leq N & \leq p + 1 + 2 \sqrt{p} \\
\text{(lower limit)} & & \text{(upper limit)}
\end{align*}
\]

1.3.2 Elliptic Curve Cryptography

The ECC can be implemented through the different values of base such as 10, 2, 6 and 3 (William Stallings 2006). The finite field over polynomial with base prime number is known as ECC over prime field ($\mathbb{Z}_p$). It is suitable for implementing software application which is briefly discussed in chapters 3 and 5. The finite field over polynomial with radix (2) is known as ECC over for $2^n$ binary field. It is more suitable for hardware application. Both are briefly discussed in the chapters 4 and 6.

Elgamal Cryptosystem is combined with EC over finite field for implementing Elliptic Curve Cryptography in a better way (Osman Ugus 2009). It consists of encryption, decryption and key generation procedures. It is introduced by Taher Elgamal which states that:

- $p$ is a large prime
- $e_1$ is a primitive root in the group $G=\langle \mathbb{Z}_p^*, \times \rangle$
- $r$ is an integer

Then, $e_2 = e_1^r \mod p$ is easy to compute, But it is infeasible to calculate $r = \log_e e_2 \mod p$ based on $e_2$, $e_1$ and $p$. 
It is mainly implemented in two different ways. They are ECC over prime field GF(p) and ECC over binary field GF($2^n$). It is more secure, when the value of p is at least 300 digits and the value of r is a few digits for each encryption to avoid low modulus attack (David JP 2007). It has the following merits to understand ECC clearly compared to other cryptosystem.

1. It is defined with multiplicative group over EC for simulation
2. Two exponents are in multiplicative group which are used as multipliers of simulation points in EC
3. The private key of sender and the secret key of receiver are integers
4. The exponentiation is replaced with the point multiplication and the point multiplication is replaced with the point addition
5. The multiplicative inverse is identified as the inverse of a point on the curve through simulation
6. The point calculation is made easy because of the following reasons:
   1. The multiplication is simpler than the exponentiation
   2. The addition is simpler than the multiplication
   3. The finding inverse is much simpler than the multiplicative
7. The strength of ECC depends upon the difficulty of solving the EC-logarithm problem
1.3.3 **Elliptic Curve Digital Signature Algorithm**

The ECDSA is specified in the document ANS X9.62 and it is approved by FIPS 186-3. It includes the requirements for obtaining the assurances of valid digital signatures. The FIPS 183 Standard defines the different methods for providing integrity, authentication and non-repudiation along with confidentiality to information through the digital signature functions. The DS functions are implemented with any of the following approaches.

1. Digital Signature Algorithm (DSA)
2. RSA based on Digital Signature Algorithm
3. Elliptic Curve based on Digital Signature Algorithm (ECDSA)

Here, the thesis is only focused on ECDSA over finite field. The main objective of ECDSA is to provide a high secure integrity, authentication and non-repudiation for information with confidentiality through the minimum level configurations (Hamed Taherdoost et al 2011). It needs only the less memory space and low power for its processing. It also generates the smaller key size for better security services. It is classified into two types based on arithmetic field. One is ECDSA over GF(p) for software application and another is ECDSA over GF(2^n) for hardware application. Both are discussed in the chapters 5 and 6 respectively.

1.4 **CODE SCHEDULING**

The primary objective of code optimization is to reduce the number of instructions reference during the execution. It automatically optimizes the execution time and memory space. So it is necessary to schedule the code for maximizing the amount of overlapping among different operations. Then the
scheduling is mainly used for implementing the program to increase the speed of execution (Dromey RG 2007). The following section describes the code scheduling and its types in detail.

1.4.1 Basics of Code Scheduling

The code scheduling is a critical issue for implementing the software applications and the designing of processors, because it should ensure the separation of dependent and independent processes before or during the execution. This will be more useful for parallel processing. The independent processes are identified and tried to execute concurrently to avoid control dependencies, data dependencies, loop carried dependencies and resource limitations. Mostly, the parallel processing is implemented through a pipelining concept. It means that the overlapping of execution is based on independent operations. This overlapping mainly depends upon the program structure of code scheduling. Today the pipelining is the key for improving the speed of the processes (Mishra Pradeep Kumar 2006).

The pipelining concept tries to reduce the number of clock cycles needed for an instruction execution, which is measured, based on the ideal pipeline of Clock Cycles Per Instruction (CPI), structural, data and control hazards. The ideal pipeline of CPI is a measure of the maximum performance attainable based on the implementations. A hazard is formed whenever there is dependence between instructions and it makes the delay for execution. This delay time in the pipelining is known as bubbles or stalls. So the stall is going to create different types of hazards on the pipeline. The examples for different hazards are: the data, instruction, structural and control hazards (Zhimin Chen et al 2011). A structural hazard is identified by the processor which tries to use the same memory for different instructions and data. A data hazard occurs when the pipeline changes the order of read/write accesses to operands. So the order differs from the sequentially executing order of instructions. The control
hazards means that the processor will not know the outcome of the branch, when it needs to insert a new instruction into the pipeline. The Table 1.4 shows some of the techniques to avoid the different hazards and stalls.

Table 1.4 Different types of techniques for avoiding various hazards on pipeline processing

<table>
<thead>
<tr>
<th>Types of Hazards</th>
<th>Techniques to avoid hazards and stalls</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ideal CPI</td>
<td>• Issuing multiple instruction per cycle</td>
</tr>
<tr>
<td></td>
<td>• Compiler dependence analysis</td>
</tr>
<tr>
<td></td>
<td>• Software pipelining</td>
</tr>
<tr>
<td></td>
<td>• Trace scheduling</td>
</tr>
<tr>
<td></td>
<td>• Hardware support for compiler speculation</td>
</tr>
<tr>
<td>Data Hazards</td>
<td>• Forwarding and bypassing</td>
</tr>
<tr>
<td></td>
<td>• Basic dynamic scheduling</td>
</tr>
<tr>
<td></td>
<td>• Dynamic scheduling with renaming</td>
</tr>
<tr>
<td></td>
<td>• Hardware speculation</td>
</tr>
<tr>
<td></td>
<td>• Dynamic memory disambiguation</td>
</tr>
<tr>
<td></td>
<td>• Basic compiler pipeline scheduling</td>
</tr>
<tr>
<td></td>
<td>• Compiler dependence analysis</td>
</tr>
<tr>
<td></td>
<td>• Software pipelining</td>
</tr>
<tr>
<td></td>
<td>• Trace scheduling</td>
</tr>
<tr>
<td></td>
<td>• Hardware support for compiler speculation</td>
</tr>
<tr>
<td>Control Hazards</td>
<td>• Delayed branches and simple branch scheduling</td>
</tr>
<tr>
<td></td>
<td>• Branch prediction</td>
</tr>
<tr>
<td></td>
<td>• Loop unrolling</td>
</tr>
<tr>
<td></td>
<td>• Hardware support for compiler speculation</td>
</tr>
</tbody>
</table>
The simplest and most common way to increase the instruction level parallelism is to exploit parallelism on different iterations in a loop. It is also known as loop level parallelism. There are a number of techniques to convert this loop level parallelism into straight line parallelism. One of the methods is the loop unrolling. It is implemented either statically or dynamically. The advantages of this code scheduling are as follows:

- Selection of one-to-one, one-to-some, and one-to-all communications
- Removal of redundant communication primitives
- Release of useless data space
- Reduction of idle communication times

The relationships among the different hazards, stalls and dependent instructions on pipeline stages during parallel computations is shown in Figure 1.16 and its corresponding remedy is given in Figure 1.17.

Figure 1.16  Different ways of affecting the parallel computation through various dependencies, hazards and stalls
Apart from the primary objective of code scheduling, the other objectives are as follows (Dromey RG 2007):

3.1. Fairness to all processes: the number of well-defined equations for all computation.

3.2. Be predictable: all computation values are properly identified before or at the time of execution.

3.3. Minimize overhead: the number of dependencies and redundancies are reduced during the processing.

3.4. Balance available resources: the variables are computed within the time and stored in appropriate variables in memory.

3.5. Enforcement of priorities: the order of execution for different computations is well defined.
3.6. Achieve balance between response and utilization: the different computation results are passed into proper place for further processing.

3.7. Maximize throughput: the execution time of computation is reduced and the speed / time complexity is optimized.

3.8. Avoid indefinite postponement and starvation: the delay time of some computations which are created by hazards are reduced.

3.9. Favor processes exhibiting desirable behavior: the computations should produce accurate results all times.

3.10. Degrade gracefully under heavy load: will find out suitable implementations to handle large amount of computations.

This code scheduling is used for implementing the different point computations on Elliptic Curve to avoid dependent operations. For example, the point multiplication of EC is computed, based on the point addition and the point doubling through the scheduling. This code scheduling is implemented in two ways. One is Static Scheduling and another is Dynamic Scheduling. The static scheduling can be done before execution (compile) time which is explained in the section 1.4.2 and the dynamic code scheduling is during the processing (run time) described in the section 1.4.3.

1.4.2 Static Scheduling for Software

The static scheduling for a software technique rearranges the code sequence before execution to reduce run-time delays. It is also known as early code scheduling. This code scheduling minimizes the number of actual hazards and resultant stalls at compile time. This scheduling requires intelligent compilation support for execution. So it is mainly dependent on user intelligence. It is also assisted with hardware interlocking to enforce the
correctness of execution (Hennessy JL et al 2007). Besides, it is classified into local or global scheduling.

The local scheduling states that the instructions are statements which cannot move from block boundaries. So it is also called basic block scheduling. The branch and basic block scheduling are examples for local scheduling.

1. Branch scheduling: It is used to find out the branch ‘is taken’ or ‘not taken’ for computing the branch target.

2. Basic-block scheduling: It is used to find out data dependencies within a block. It is also called list scheduling.

The global scheduling means that the instructions are moving from one block to another block. Some data is common for two or more blocks. The cross block, software pipelining and trace scheduling are examples for this type of scheduling.

2.1. Trace scheduling: Trace scheduling is a control scheduling which tries to optimize the flow path that is executed frequently.

2.2. Percolation scheduling: The role of this scheduling is to perform the necessary code motion based on instruction scheduling. It must be developed to allow one instruction to be moved ahead of another instruction.

2.3. Software Pipelining: It is a method for increasing the parallelism of instruction scheduling based on data dependencies. It also limits the opportunity for parallel processing. It can overlap the loop iterations for scheduling among dependencies.
2.4. Cross Block Scheduling: It improves on basic block scheduling by considering a tree of blocks at once and may move instructions or statements from one block to another block.

This software code scheduling is used for implementing point multiplication of EC over prime field to reduce the number of hazards before the execution. It is more useful for parallel computation. Many machines are pipelined and exposed to some aspects of pipelining for user such as a branch delay slots, memory-access delays and multi-cycle operations. So it is important to analyze dependent and independent processes for reducing the overlapping operations. Then, it will automatically minimize the number of dependencies, hazards and stalls between the point operations.

1.4.3 Dynamic Scheduling for Hardware

Another approach is a dynamic scheduling for the processors, where the instructions are rearranged for hardware execution to avoid the hazards during the execution. It is also known as late code scheduling. This late code scheduling requires sophisticated hardware support. Its decision can have a major impact on the performance of parallel computation. In practice, the dynamic scheduling is assisted with static scheduling to improve performance and tried to reduce hardware cost (Hennessy JL et al 2007). The main advantages of dynamic scheduling are as follows:

- Performance portability
- Do not want to recompile for new machines
- More information available
- Memory addresses, branch directions
- More registers available
• Compiler may not have enough to schedule well
• Speculative memory operation re-ordering
• Compiler must be conservative & hardware can speculate
• But compiler has a larger scope
• Compiler does as much as it can (not much)
• Hardware does the rest

A major limitation of the pipelining techniques is an in-order instruction issue. If an instruction is stalled in the pipeline, the instructions will not proceed. It means that if there is a dependency between two closely spaced instructions in the pipeline, then it will stall. This scheduling is used for implementing the point multiplication over EC of binary field. It is very much useful for parallel computation. During this processing, there is no possibility for occurring warnings and errors, because it is going to avoid the number of dependencies, hazards and stalls.

1.5 PROBLEM DOMAIN, OBJECTIVES AND MOTIVATION

The problem definition is that the code scheduling is used for implementing different point computations on Elliptic Curve to identify dependent processing and convert into independent processing especially on point multiplication of EC.

It is classified into two types such as static code scheduling and dynamic code scheduling. The static code scheduling can be done at compile time for software application and the dynamic code scheduling at run time for hardware application.
The main objective is that the code scheduling is used for implementing point computations especially on point addition and point doubling in EC over Finite Field to compute point multiplication.

The motivation is to analyze the dependent and independent processes of point computations for avoiding overlap in parallel processing, then it tries to minimize or avoids number of dependencies, hazards and stalls among the points of EC over Finite Field and increase possibility for parallel computation. So the execution speed and the life time of hardware are increased significantly. For better understanding, the thesis is organized into seven chapters and the road map of this thesis is given in Figure 1.18.

Figure 1.18 Road map of the thesis