CHAPTER 3

ECC OVER GF(p) WITH STATIC SCHEDULING

This chapter suggests an alternative approach for calculating the point multiplication of Elliptic Curve Cryptography (ECC) over Prime Field GF(P) for parallel computations. It also explains one of the applications known as the Crypto Editor for this approach.

3.1 INTRODUCTION

This chapter suggests a software code scheduling technique for Elliptic Curve Cryptography over prime field GF(p) to provide information security. The performance, security, size and versatility of ECC are mainly focused on the following parameters (Estes Mattew et al 2006).

1. Finite Field Type
2. Elliptic Curve Type
3. Point Coordinate System
4. Algorithms Nature
5. Protocol Definition
6. Key Length
7. Hardware /Software /Hardware-Software Considerations
8. Way of Implementation
9. Memory Space Usage
10. Size of Code
Based on these, the different parameters are considered to implement ECC over GF(p) as shown in Table 3.1. The important operations of ECC over GF(p) are the point addition and point multiplication. They are scheduled in appropriate format through coding to support the parallel processing. Here, The point addition is an atomic operation. It does not need any scheduling for optimizing the speed of computation. But the point multiplication is a complex operation which needs more time to complete its operation (Erkay Savas et al 2005). So it is necessary to confirm that the point multiplication is mathematically secured. Already a lot of mathematical proofs are available to test the strength of point multiplication (Jean-Claude Bajard et al 2006). But today, there is no proper optimized technique for testing this operation in the way of parallel computation.

**Table 3.1 Different parameters considered for ECC over prime field GF(p) implementation**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>ECC over GF(P) Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finite Field</td>
<td>Based on the prime number P</td>
</tr>
<tr>
<td>Elliptic Curve</td>
<td>Any type</td>
</tr>
<tr>
<td>Point Coordinate System</td>
<td>Any type</td>
</tr>
<tr>
<td>Algorithms</td>
<td>Any type</td>
</tr>
<tr>
<td>Protocol Definition</td>
<td>Any type</td>
</tr>
<tr>
<td>Key Length</td>
<td>Any Size</td>
</tr>
<tr>
<td>Hardware/Software/both</td>
<td>Software</td>
</tr>
<tr>
<td>Way of Implementation</td>
<td>To support Parallel computation</td>
</tr>
<tr>
<td>Memory Space Usage</td>
<td>Optimum Level</td>
</tr>
<tr>
<td>Size of Code</td>
<td>Feasibility Level</td>
</tr>
</tbody>
</table>
Some of the examples for ECC over prime field GF(p) are P-128, P-163, P-192, P-224, P-256, P-384 and P-521, where P is a prime number and the number denotes the number of bits needed to define the prime (Branovic I et al 2003). Here, it is modified by using code scheduling for software applications. This code scheduling reduces the number of dependent operations to create the possibility for parallel computation. So the execution time is reduced and the speed of algorithm is automatically increased (Hai Yan et al 2006).

For these reasons, this chapter has been organized into seven sections to explain in detail. The overview of ECC over GF(p) is discussed briefly in the section-3.2. Next, the section-3.3 proposes a code scheduling for the point multiplication of ECC over GF(p) on software applications. It is followed by the section-3.4 which indicates the result of point multiplication based on linear and proposed implementations. Subsequently, the section-3.5 analyzes the result of the proposed methodology in different perspective. Before concluding the section-3.7, the section-3.6 discusses one of the applications of the proposed work with an example.

3.2 OVERVIEW OF ECC OVER GF(p)

The proposed methodology is a combination of ECC over GF(p) and code scheduling for software applications. It is also known as static scheduling. The procedures of each concept in ECC over GF(p) are described in Section 3.2.1 and the static code scheduling of this method is explained in Section 3.2.2.

3.2.1 ECC over Prime Field

The Elliptic Curve Cryptography over prime field GF(p) is simulated based on the concept of ElGamal algorithm, finite field and
polynomial. It has three procedures called by key generation, encryption and decryption, which are described in the following sections (Sangook Moon et al 2001).

3.2.1.1 Key generation

The key generation is a set of rules for the Elliptic Curve constraints to produce a pair of private and public keys. These keys are distributed to corresponding users to generate a secret key for communication. The key generation needs some global parameters for finding a key pair. The private key is assumed as an integer and it is chosen based on one of the global parameters. It is kept as secret key and the public key is generated based on these values (Forouzan BA 2008). The procedure of key generation is as follows:

Common parameters

1. Define \( E_q(a,b) \) where \( a, b \rightarrow \text{EC parameters and } q \rightarrow \text{prime number} \).
2. Find out \( n \times G = 0 \) where \( G \) is point on EC whose order is large value of \( n \)

A pair of key generation on sender side

1. Select a private key \( n_A \) where \( n_A < n \)
2. Calculate a public key \( P_A \) where \( P_A = n_A \times G \)

A pair of key generation on receiver side

1. Select a private key \( n_B \) where \( n_B < n \)
2. Calculate a public key \( P_B \) where \( P_B = n_B \times G \)
Finding a secret key on sender side

Calculate the secret key on sender side \( K = n_A \times P_B \)

Finding a secret key on receiver side

Calculate the secret key on receiver side \( K = n_B \times P_A \).

Verification and Validation for generating secret key

The secret key is computed based on its own private key with other person public key and it is verified as follows.

(Secret key-one side): \( K = n_A \times P_B = n_A \times n_B \times G \)

\[ = n_B \times n_A \times G = n_B \times P_A = K \] (Secret key-another side)

3.2.1.2 Encryption

The collections of characters in the plain text are converted into a collection of points through mapping technique. The mapping is one to one correspondence between character symbol and the point on EC. It is a challenging issue for simulating EC points (Osman Ugus et al 2009). Here, an encryption converts the point value of plain text into the point value of cipher text through the point manipulation by using the secret key. The cipher text has two points \( C_1 \) and \( C_2 \) based on Elgamal algorithm.

It is denoted by,

\[ \text{Cipher Text} = (C_1, C_2) \]

where \( C_1 = kG, C_2 = \text{Plain Text} + kP_B \), and

\( k = \text{a large random positive integer} \)
3.2.1.3 Decryption

The decryption converts the point value of cipher text into the point value of plain text through the same point manipulation by using same secret key. After, the point value of plain text is converted into the plain text value through the same mapping technique (William Stallings 2006).

It is denoted by,

\[
\text{Plain Text} = (C_2 - dC_1)
\]
\[
= (\text{Plain Text} + kP - nB(kG))
\]
\[
= (\text{Plain Text} + k(n_B \times G) - nB(kG))
\]
\[
= (\text{Plain Text} + 'O')
\]
\[
= \text{Plain Text}
\]

The following example is considered to describe encryption, decryption and key generation of ECC over prime field.

**Common parameters**

1. Define \(E_q(a,b) = E_{67}(2,3)\) where \(a=2\), \(b=3\) and \(p=67\)
2. Assume \(G=(2,22)\)

**A pair of key generation on sender side**

1. private key \(n_A=4\)
2. public key \(P_A = n_A \times G \mod p = 4 \times (2,22) \mod 67 = (13,45)\)
A pair of key generation on receiver side

1. private key $n_B = 2$
2. public key $P_B = n_B \times G \mod p = 2 \times (2,22) \mod 67 = (35,1)$

Encryption

The plain text is assumed as $P = (24,26)$, which is derived through mapping technique based on the American Standard Code for Information Interchange (ASCII) values and a prime number. The point value of cipher text is computed as follows:

\[ C_1 = kG = 2 \times (2,22) = (35,1) \mod 67 \]
\[ C_2 = \text{Plain Text} + kP_B = \text{Plain Text} + 2 \times (31,1) \mod 67 \]
\[ C_2 = (24,26) + 2 \times (31,1) \mod 67 = (23,25) \]

Therefore, the value of cipher Text is the combination of $C_1$ and $C_2$.

Decryption

The point value of plain text is regenerated from the point value of cipher text based on the following equation.

\[ \text{Plain text point} = (C_2 - dC_1) \mod p = (23,44) - 2 \times (35,1) \mod 67 \]
\[ = (21,44) - (23,25) \mod 67 = (24,26) \]

This point value is converted into a plain text symbol based on the same mapping technique.
3.2.2 Software Code Scheduling

The point multiplication is playing the major role in ECC over prime field. So it is necessary to optimize the execution time of this operation. The code scheduling is suggested for this operation with a divide and conquer strategy to increase the speed. The divide and conquer is a process to break down a problem into two or more sub-problems. Finally, the solutions of these sub-problems are combined together to find out the solution for the original problem. Here, it is tried to solve the sub problems concurrently for parallel processing (Mishra Pradeep Kumar 2006).

The divide and conquer also converts the number of dependencies in the point multiplication into the number of independencies of point operations. These operations are computed by using point addition and point doubling separately, and finally combined together to find out the original value of point multiplication.

3.3 PROPOSED METHODOLOGY FOR ECC OVER GF(p)

Based on Equation (2.1), the binary and skew trees are created with the help of ‘k’ to compute the point multiplication. The binary trees are used to perform the point doubling and the skew trees for the point addition. Each node in trees has a point value (P) which may be computed through the point addition or point doubling. Then, the summing of skew tree and binary tree values will compute the final value of point multiplication (kP).

For example, the k value is assumed as 15, when it is divided by 2 in each time, the quotient value becomes 7, 3, 1 and the remainder values are 1, 1, 1. The binary trees for point doubling are created based on the quotient values. The node has a point value and it looks like a complete binary tree as shown in Figure 3.1.
Figure 3.1  Point doubling of $kP$, where $k$ is assumed as even number and $P$ as a point value

Here, point doubling is performed based on Equation (1.6). The algorithm of a point doubling operation based on the value of $k$ is as follows:

**Algorithm :Point Doubling Operations of $kP$ based on even number.**

Input  : Point $P(x,y)$ //Point
        : a scalar value $Q$ //Quotient
Output: $QP$

Point Sum(0,0);
Sum=$P$
if (P(x,y)== Sum(0, 0)) then
  Sum (x,y)= P(x,y)
else if (P(x,y)== Sum(x,-y)) then
  Sum (x,y)= P(x,0);
else if(P(x,y)==Sum(x,y)) then
  Sum (x,y)=P(x,y)+ Sum (x,y) // Point Doubling
else
  Sum (x,y)=P(x,y)+ Sum(x,y) // Point Addition and
  // (P(x,y)!=Sum (x,y))
end if
return SUM.
Subsequently, the skew trees for point additions are calculated based on the remainder and point values. It forms incomplete binary trees as shown in Figure 3.2. These trees are also known as skew tree computation.

![Skew Tree Diagram](image)

**Figure 3.2**  Point addition of kP, where the value of k is odd number and p is a point value

This point addition is computed based on Equation (1.7). The algorithm of a point addition operation based on the value of k is as follows:

**Algorithm : Point Addition Operations of kP**

Input: Point P, a scalar value R //remainder
Output: RP
Point Sum=(0,0);
if (P(x,y)== Sum(0, 0)) then
    Sum (x,y);= P(x,y);
else if(P(x,y)== Sum(x,-y)) then
    Sum (x,y)= P(x,0);
else if(P(x,y)==Sum(x,y)) then
    Sum (x,y)=P(x,y)+ Sum (x,y) // Point Doubling
else if
    Sum (x,y)=P(x,y)+ Sum(x,y) // Point Addition
end if
return SUM
Finally, these two trees are summed up by using any of Equations (1.4), (1.5), (1.6) or (1.7) to compute the value of point multiplication (kP), which are shown in the following Figures 3.3 and 3.4.

![Diagram](image)

**Figure 3.3** Point computation of kP for dividing processes, where k = 15 and P = point value

The algorithm for a point multiplication of kP based on the value of k through the dividing operation is as follows:

**Algorithm : Point Multiplication of kP based on dividing operations**

- **Input** : a scalar value k,
- Point P(x,y)
- **Output**: kp
- Point P1=(0,0)
- Point P2=(0,0)
- If k=1 then
  - P=P
repeat
Q←k/2;
if(Q>0) then
   \( P_1=\text{call PointDoublingBinary(Point P)} \)
   \( P=P_1 \)
end if
R←k mod 2
if(R=1) then
   \( P_2=\text{call PointDoublingSkew(Point P)} \)
   \( P=P_2 \)
end if
k=k/2
until (k>1)
call Procedure for Point Summazation (Point P_1, Point P_2)

Figure 3.4 Point computation of kP for conquering processes, where k = 15 and P = a point value
The algorithm for point multiplication based on the value of k through conquering processes is as follows:

**Algorithm : Point Summarization of kP**

Input: Point $P_1$, Point $P_2$

Output: $kP$ is summation of $P_1$ and $P_2$

if $(P_1(x,0) == P_2(x,y))$ then

$\text{Sum} (x,y) = P_1(x,y)$

else if($P_1(x,y)==P_2(x,0)$) then

$\text{Sum} (x,y) = P_2(x,y)$

else if $(P_1(x,y) == P_2(x,-y))$ then

$\text{Sum} (x,y) = P_1(x,0)$

else if($P_1(x,-y)== P_2(x,y)$) then

$\text{Sum} (x,y) = P_2(x,0)$

else if($P_1(x,y) == P_2(x,y)$) then

$\text{Sum} (x,y) = P_1(x,y) + P_2(x,y)$

else

// $(P_1(x,y)!== P_2(x,y))$

$\text{Sum} (x,y) = P_1(x,y) + P_2(x,y)$

end if

Display SUM

This innovative point multiplication methodology minimizes the number of loop carried dependence. The numbers of computations in the iterations are determined based on binary and skew tree computations. They
are reduced further through code scheduling. Here, the data dependencies are classified into two categories. One is the point doubling dependencies and another is point addition dependencies. Next, the control dependencies are scheduled, based on the point doubling and point addition computation to the next level passing of computation. These dependencies are automatically to create the different types of hazards and stalls. So these are avoided for maximum through the code scheduling.

### 3.4 RESULT

The study of mathematical theories describes the resources, which are required by a computing machine to solve the problem known as computational complexity theory. It is important for both the theoretical and practical of computer science especially in cryptography. The computational complexity of point multiplication is analyzed, based on the time complexity in three different ways. They are: the best, worst and average cases of time complexities. Here, it is analyzed based on the execution time in seconds. It is implemented with ‘C’ programming to measure the execution time. For example, the following configurations of the computing machine are considered to implement for the point multiplication.

- **Processor name**: Intel ® core™
- **Number of Processor**: 4
- **Processor type**: Quadrable core with a die size of 164 mm²
- **Processor Speed**: 2.33 Giga Hertz
- **Processor space**: 2 Giga bits
- **Word length of**
- **Operating system**: 64 bits
The following equation is considered for this proposed methodology and the various values of x and y are substituted in this equation to find out a set of points on Elliptic Curve.

\[ y^2 = x^3 + ax + b \]  

(3.1)

The equation \( E_p(a,b) \) is assumed as \( p=11111, \ a=1 \) and \( b=1 \). It is denoted as follows:

\[ E_p(a,b) = E_{11111}(1,1) \]  

(3.2)

Then, a point \( (x=0, y=1) \) is taken from this set to compute the point multiplication for both the linear and the proposed point multiplications as shown in Table 3.2. During the processing, the exertion time of proposed and linear point multiplications are measured in terms of seconds. These execution times are analyzed based on Amdahl's Law and processor’s performance equations which are discussed in appendix 1 and appendix 2 respectively.

**Table 3.2 Different parameter value needed for simulating point multiplication of \( kP \)**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( E_p )</td>
<td>Input</td>
<td>( E_{11111} )</td>
</tr>
<tr>
<td>a</td>
<td>Input</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>Input</td>
<td>1</td>
</tr>
<tr>
<td>x</td>
<td>Input</td>
<td>0</td>
</tr>
<tr>
<td>y</td>
<td>Input</td>
<td>1</td>
</tr>
<tr>
<td>P</td>
<td>Input</td>
<td>( (x,y) )</td>
</tr>
<tr>
<td>k</td>
<td>Input</td>
<td>Number of times(N)</td>
</tr>
<tr>
<td>( k_i )</td>
<td>Input</td>
<td>( 0 &lt; N &lt; p )</td>
</tr>
<tr>
<td>( kP )</td>
<td>Output</td>
<td>Point Multiplication Value</td>
</tr>
<tr>
<td>( kP ) execution time</td>
<td>Output</td>
<td>Seconds</td>
</tr>
</tbody>
</table>
The best case of time complexity is to find out the optimal cases of execution for point multiplication. In the proposed point multiplication, there is no remainder for the value of ‘k’ in all iterations. It means that $k = 2^n$, where $0 < N$ and it is even number. So it takes only $\log_2 N$ times to compute $kP$ based on quotient value and a point. There is no need to compute the skew tree point multiplication. Because the remainder value of point computation is zero. So the point multiplication is computed only through the point addition. But the linear point multiplication takes $N-1$ times to compute $kP$ based on $N$ where $k = N$. The comparisons between these two cases are shown in Table 3.3 to analyze point multiplications.

Table 3.3 Execution time of linear multiplication and the proposed tree computation of best case in seconds

<table>
<thead>
<tr>
<th>Compute kP</th>
<th>Time in seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i \ (1 \ TO \ K)$</td>
<td>$2^i$</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
</tr>
<tr>
<td>6</td>
<td>64</td>
</tr>
<tr>
<td>7</td>
<td>128</td>
</tr>
<tr>
<td>8</td>
<td>256</td>
</tr>
<tr>
<td>9</td>
<td>512</td>
</tr>
<tr>
<td>10</td>
<td>1024</td>
</tr>
<tr>
<td>11</td>
<td>2048</td>
</tr>
<tr>
<td>12</td>
<td>4096</td>
</tr>
<tr>
<td>13</td>
<td>8192</td>
</tr>
</tbody>
</table>
The second case is a worst case. It defines the way of algorithm running all possible conditions. In the proposed point multiplication, there is a remainder and quotient values for the value of ‘k’ in all iterations, when it is an odd number. The value of ‘k’ takes only $2\log_2 N$ times to compute point doubling and the point addition based on the quotient and remainder values. So the time complexity of this is measured as the summation of $O(\log_2 N)$ for point doubling and $O(\log_2 N)$ for point addition. So the total time complexity of point multiplication is defined by $O(\log_2 N + \log_2 N)$. In linear point multiplication, it takes $N-1$ times to compute $kP$ based on $N$ where $k=N$. So there is no tremendous change in the time complexity. The comparisons between these two cases are shown in Table 3.4.

Table 3.4 Execution time of linear multiplication and the proposed tree computation of worst case in seconds

<table>
<thead>
<tr>
<th>i (1 TO K)</th>
<th>$2^i-1$</th>
<th>Compute kP</th>
<th>Time in seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>i (1 TO K)</td>
<td>$2^i-1$</td>
<td>linear (sec)</td>
<td>worst (sec)</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>0.165</td>
<td>0.055</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>0.22</td>
<td>0.165</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>0.33</td>
<td>0.275</td>
</tr>
<tr>
<td>5</td>
<td>31</td>
<td>0.714</td>
<td>0.33</td>
</tr>
<tr>
<td>6</td>
<td>63</td>
<td>1.319</td>
<td>0.385</td>
</tr>
<tr>
<td>7</td>
<td>127</td>
<td>2.527</td>
<td>0.495</td>
</tr>
<tr>
<td>8</td>
<td>255</td>
<td>5</td>
<td>0.55</td>
</tr>
<tr>
<td>9</td>
<td>511</td>
<td>10.495</td>
<td>0.604</td>
</tr>
<tr>
<td>10</td>
<td>1023</td>
<td>20.879</td>
<td>0.604</td>
</tr>
<tr>
<td>11</td>
<td>2047</td>
<td>41.648</td>
<td>0.659</td>
</tr>
<tr>
<td>12</td>
<td>4095</td>
<td>82.583</td>
<td>0.714</td>
</tr>
<tr>
<td>13</td>
<td>8191</td>
<td>167.802</td>
<td>0.769</td>
</tr>
</tbody>
</table>
The third case is an average case, in which the algorithm acts under some probabilities of execution. Here, there is a remainder value for \( k \) in some cases and no remainder for some other cases. Therefore, the computation time of this case is measured by \( \log_2 N \) for point doubling and Probability of \( \log_2 N \) times for point addition. So it is denoted by \( O(\log_2 N) + O(\text{Probability of } \log_2 N) \). All these time complexities are redefined by \( \log_2 N + 1, \ 2\log_2 N + 1 \) and \( \log_2 N + \text{Prob}\{\log_2 N\} + 1 \), where the value ‘1’ denotes the final computation of \( kP \) based on combining tree values.

### 3.5 PERFORMANCE ANALYSIS

The proposed and linear point multiplications are analyzed with the following graphs shown in Figures 3.5, 3.6, 3.7 and 3.8. In these graphs, the x-axis denotes the value of \( k \) in the form of \( 2^i \) and the y axis denotes the number of seconds. The values are taken from Tables 3.3 and 3.4 to plot the different graphs.

The first graph as shown in Figure 3.5 illustrates the number of seconds needed to compute point multiplication of \( kP \) for the different values of \( k \). It is analyzed with 12 samples such as \( 2P, 4P, 8P, 16P, 32P, 64P, 128P, 256P, 512P, 1024P, 2048P \) and \( 4096P \) where \( P \) is a point \((0,1)\). Then the corresponding execution time of point multiplication are 0.165s, 0.22s, 0.33s, 0.714s, 1.319s, 2.527s, 5s, 10.495s, 20.879s, 41.648s, 82.583s and 167.802s.
Figure 3.5  Total amount of execution time needed to compute kP for Liner point multiplication in this graph

Next, the second graph shown in Figure 3.6 explains the number of seconds needed to compute point multiplication kP for the various values of k in the form of $2^i$ or $2^i-1$. The first form of $2^i$ illustrates the best case time complexity of point multiplication denoted as blue colour curve, and $2^i - 1$ for the worst time complexity of point multiplication denoted as red colour curve in the graph. It is also analyzed with 12 samples for both point multiplications.

The 2P, 4P, 8P, 16P, 32P, 64P, 128P, 256P, 512P, 1024P, 2048P and 4096P are considered for the best case time complexities where P is a point (0,1). The corresponding execution times are as follows: 0.055s, 0.11s, 0.165s, 0.22s, 0.275s, 0.33s, 0.33s, 0.385s, 0.385s, 0.44s, 0.495s and 0.495s. The P, 3P, 7P, 15P, 31P, 63P, 127P, 255P, 511P, 1023P, 2047P and 4095P are considered for the worst case time complexities where P is the same as before. The corresponding execution times of this case are 0.055s, 0.165s, 0.275s, 0.33s, 0.385s, 0.495s, 0.55s, 0.604s, 0.604s, 0.659s, 0.714s and 0.769s.
Figure 3.6  Total amount of time needed to compute kp based on best & worst cases of point multiplication in this graph

The third graph in Figure 3.7 shows the comparison between the worst case of time complexity in the proposed point multiplication and the linear point multiplication. The linear point multiplication is shown in blue colour on the graph denotes the large value of k which takes more time for execution. When the recent value of point multiplication is compared with the previous value of point multiplication (kP), it takes twice the time of execution for the next level. But the proposed point multiplication in the worst case, there is a minimum gap between the current and previous computations shown in red colour on the graph.
Figure 3.7  Numbers of clock pulses needed to compute kP by using the linear scalar with the worst case point multiplication in this graph

The fourth graph in Figure 3.8 shows the comparison between the best case of point multiplication shown as red color line and linear point multiplication shown as blue colour in the graph. Here, the time gap of the best case in the point multiplication is measured in the level of constant amount of times.

Figure 3.8  Numbers of clock pulses needed to compute kP by using the linear scalar with the best case point multiplication in this graph
This final graph shown in Figure 3.9 illustrates the comparisons among the different ways of proposed point multiplication with the linear point multiplication. The best case time complexity of point multiplication is compared with the worst case time complexity and it is always lesser in value. Next, these lines are compared with linear case and they always produce better results.

![Graph illustrating comparisons among different ways of proposed point multiplication with linear point multiplication.](image)

**Figure 3.9** Compares the numbers of clock pulses needed to compute \(kP\) by using the linear scalar with the best case point multiplication in this graph.
Table 3.5 Comparative study of the proposed point multiplication with the linear point multiplication

<table>
<thead>
<tr>
<th>The value of ‘k’</th>
<th>Linear Point Multiplication (sec)</th>
<th>Proposed point Multiplication (sec)</th>
<th>The percentage of improvement (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 (best)</td>
<td>0.33</td>
<td>0.165</td>
<td>50%</td>
</tr>
<tr>
<td>15 (worst)</td>
<td>0.33</td>
<td>0.275</td>
<td>17%</td>
</tr>
<tr>
<td>256 (best)</td>
<td>5</td>
<td>0.33</td>
<td>93%</td>
</tr>
<tr>
<td>255 (worst)</td>
<td>5</td>
<td>0.55</td>
<td>89%</td>
</tr>
<tr>
<td>4095 (best)</td>
<td>82.583</td>
<td>0.495</td>
<td>99.4%</td>
</tr>
<tr>
<td>4096 (worst)</td>
<td>82.583</td>
<td>0.714</td>
<td>99.1%</td>
</tr>
</tbody>
</table>

When it is considering the execution time from Tables 3.3 and 3.4, the proposed methodology is better than linear point multiplication. The comparison is shown in Table 3.5.

The Double and Add, Montgomery and Joabosin are also the same level of improvement, when it compares with linear point multiplication. But these methodologies have the different dependencies to affect parallel computations. The Table 3.6 shows the comparison among different point multiplication with proposed point multiplication based on the dependencies, hazards and stalls. It will also reduce the utility of hardware in the minimum level.
Table 3.6  Comparison among the different point multiplication with the proposed tree computation

<table>
<thead>
<tr>
<th>Methodologies</th>
<th>Linear</th>
<th>Montgomery</th>
<th>Jacobian</th>
<th>Tree Multiplication</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Binary length of k value in kP denoted by n</strong></td>
<td>Best case</td>
<td>n times of point doubling</td>
<td>n times of point doubling without inversion operation</td>
<td>n times of point doubling</td>
</tr>
<tr>
<td></td>
<td>worst case</td>
<td>$2^n$ times of Point addition</td>
<td>n times of point doubling + $n$ times of point addition</td>
<td>n times of point doubling + $n$ times of point addition</td>
</tr>
<tr>
<td><strong>Number of Dependences</strong></td>
<td>data more</td>
<td>More</td>
<td>more</td>
<td>minimized</td>
</tr>
<tr>
<td></td>
<td>control more</td>
<td>More</td>
<td>less</td>
<td>minimized</td>
</tr>
<tr>
<td></td>
<td>loop carried more</td>
<td>More</td>
<td>less</td>
<td>minimized</td>
</tr>
<tr>
<td></td>
<td>name less</td>
<td>Less</td>
<td>more</td>
<td>minimized</td>
</tr>
<tr>
<td><strong>Occurrence of Hazards and Stalls</strong></td>
<td>RAW YES</td>
<td>YES</td>
<td>YES</td>
<td>minimized</td>
</tr>
<tr>
<td></td>
<td>WAR YES</td>
<td>YES</td>
<td>YES</td>
<td>minimized</td>
</tr>
<tr>
<td></td>
<td>WAW No</td>
<td>No</td>
<td>YES</td>
<td>minimized</td>
</tr>
<tr>
<td><strong>Parallel Processing</strong></td>
<td>not possible</td>
<td>not Possible</td>
<td>not possible</td>
<td>possible</td>
</tr>
</tbody>
</table>
The data dependencies in the Double and Add, Montgomery or Jacobian cases are predicted based on point addition with point doubling or point doubling with point addition, whereas the proposed point multiplication the point addition and point doubling are computed separately. The control dependencies in Double and Add are found based on the value of each binary bit to perform point addition. In the Montgomery case, the control dependencies are also depends upon on the value of each binary bit to perform point addition with point doubling or point doubling with point addition. Any of these methods is used to find out Jacobian control dependencies. But the proposed point multiplication to handle control dependencies separately to avoid the lost control dependency in the iteration.

The loop carried dependencies in the Double and Add, Montgomery or Jacobian cases are identified based on the current iteration of point computations with the next iteration of point computations. In the proposed point multiplication, the earlier or late iteration of point computations are avoided. The name dependencies are identified through the point addition and point doubling which maintains the results in the same name. It is a same number of dependencies for all point multiplication methodologies except proposed point multiplication. It includes an additional variable for maintain intermediate results.

For example, the following samples are taken from Tables 3.3 and 3.4 to compare the proposed methodology with Double and Add, Montgomery and Jacobian methodologies shown in Table 3.7.
Table 3.7 Comparison among the different point multiplication with the proposed tree computation

<table>
<thead>
<tr>
<th>Number of bits</th>
<th>Name of Dependency</th>
<th>Number of Dependencies</th>
<th>Double and Add</th>
<th>Montgomery</th>
<th>Jacobian</th>
<th>Proposed Multiplication</th>
</tr>
</thead>
<tbody>
<tr>
<td>4, 8, 12</td>
<td>Data</td>
<td></td>
<td>7, 15, 23</td>
<td>7, 15, 23</td>
<td>7, 15, 23</td>
<td>4, 8, 12</td>
</tr>
<tr>
<td></td>
<td>Control</td>
<td></td>
<td>4, 8, 12</td>
<td>8, 16, 24</td>
<td>(4, 8, 12) or (8, 16, 24)</td>
<td>3, 7, 11</td>
</tr>
<tr>
<td></td>
<td>Loop carried</td>
<td></td>
<td>3, 7, 11</td>
<td>3, 7, 11</td>
<td>3, 7, 11</td>
<td>2, 6, 10</td>
</tr>
<tr>
<td></td>
<td>Name</td>
<td></td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

From Table 3.7, when the proposed point multiplication is compared to other point multiplication, the following percentages of dependencies are reduced as in Figure 3.10.

1. 48% to 43% amount of data dependencies are reduced.
2. 25% to 8% amount of control dependencies are reduced.
3. 33% to 9% amount of loop carried dependencies are reduced.
Figure 3.10 The number of dependencies are reduced in this proposed point multiplication

3.6 DISCUSSION AND APPLICATION

The ECC over finite field is particularly beneficial for the following applications:

1. Limited power computation place such as wireless devices and personal Computer (PC) cards
2. Limited Integrated Circuit space
3. High speed requirement
4. The intensive use of the signing, verifying and authentication place
5. The signed messages storing and transmitting place for short messages
6. Limited bandwidth place such as wireless communications

Especially, the ECC over prime field is used for various applications such as sending secure information, e-Transactions and Digital Signature for Smart Cards (Malhotra K et al 2007). This section explains one
of the applications, which avoids the passive attacks on e-Mail to prevent the advertisements based on the message contents from yahoo, gmail or some other mail server (Chia-Long WU 2005). It is also used for cryptanalysis to test the strength of the algorithm.

3.6.1 Cryptanalysis of ECC over prime field

The cryptography is analyzed in two ways known as the brute force analysis and cryptanalysis techniques. These techniques are used to find out the strength of algorithm based on speed, memory space, level of security, and hardness of the problem. They are also used to analyze the relationships among the cipher text, plain text and key values (Lauter K et al 2007). To break the relationship of the brute force analysis is done by trying all combination of keys. But it is a time consuming process. But the cryptanalysis takes less time to break the relationships (Honggeng Hu et al 2007). So it needs more and additional information. Here, it is carried out through the proposed point multiplication.

3.6.2 Virtual Crypto Editor

Most of e-Mail providers are following their own standards to provide for storing, transferring and retrieving information for e-Mail users. They are promising as 100% confidentiality for information, but mostly they are not following their ethics in all cases. They allow third parties to advertise their products based on user content in thee-Mail. These types of activities are called passive attacks. It can be avoided by including a special text editor between user and e-Mail editor. This type of text editor is known as crypto editor. Particularly it is more important for mobile applications and wireless networks (Sakiyama K et al 2007), because of the minimum length of key size, less power consumption and low band width (Challa Narasimham et al 2008). So the crypto editor is suggested to implement with the proposed point
multiplication of ECC over prime field. The functionality of text editor is shown in Figure 3.11 to understand the operations of both sender and receiver sides.

**Sender side:**

1. The user types information on special text editor.
2. Then the entire information is encrypted using the proposed methodology.
3. Finally it is embedded into e-mail provider’s text editor.

**Receiver side:**

1. The user retrieves the entire information from e-mail provider’s text editor.
2. This entire information is decrypted by using the proposed methodology.
3. After this, it is displayed on separate text editor.

![Figure 3.11 Block diagram of e-Mail transfer with crypto editor](image-url)
Today, this crypto editor is optimized for the use of speed and portability. The symmetric key algorithm based crypto editor performs with quick processing. But the security level of this algorithm is lesser than the asymmetric key algorithm. When the crypto editor is implemented for software applications, the following properties must be ensured:

1. user-friendly
2. no special installation
3. simple and secure
4. high user interface
5. platform independent
6. speed
7. portability

The crypto editor over GF (p) is implemented with the help of Java Development Tool Kit (JDK), JavaScript and Hyper Text Markup Language (HTML). Theoretically, the different versions of ECC: ECC-256, ECC-384 and ECC-512 are equal to the different versions of AES: AES-128, AES-192 and AES-256 to provide the same level of security for information. Today, the recent crypto editor is implemented with only AES-256. But this chapter suggests an ECC-512 for implementation in place of AES-256 to support more confidentiality.

3.7 SUMMARY

The traditional ECC uses a linear point multiplication to perform encryption or decryption. The consequence of this approach needs more clock pulses to complete its processing. But the proposed work reduces the number of clock pulses and power consumption. And also it increases the performance of ECC. This computation is very much useful, when it is trying
to enhance secure e-Mail applications such as Pretty Good Privacy (PGP) or Secure/Multipurpose Internet Mail Extensions (S/MIME), Short Message Service (SMS), Security Protocol Design, e-Business and e-Banking. So this chapter concludes to suggest ECC over prime field for performing high speed encryption/decryption through parallel computation for software applications, for example, a virtual crypt editor.

And also, the suggested methodology supports the rapid use of point multiplication in ECC over prime field for parallel computation to reduce the power and utilizes the hardware in minimum level.