Increasing the Robustness of Point Operations in Co-Z Arithmetic against Side-Channel Attacks

by

Ziyad Mohammed Almohaimeed
B.Sc., Qassim University, 2009

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Applied Science

in the Department of Electrical and Computer Engineering

Increasing the Robustness of Point Operations in Co-Z Arithmetic against Side-Channel Attacks

by

Ziyad Mohammed Almohaimeed
B.Sc., Qassim University, 2009

Supervisory Committee

Dr. Mihai Sima, Supervisor
(Department of Electrical and Computer Engineering)

Dr. Michael L. McGuire, Departmental Member
(Department of Electrical and Computer Engineering)

Supervisory Committee

Dr. Mihai Sima, Supervisor
(Department of Electrical and Computer Engineering)

Dr. Michael L. McGuire, Departmental Member
(Department of Electrical and Computer Engineering)

ABSTRACT

Elliptic curve cryptography (ECC) has played a significant role on secure devices since it was introduced by Koblitz and Miller more than three decades ago. The great demand for ECC is created by its shorter key length while it provides an equivalent security level in comparison to previously introduced public-key cryptosystems (e.g. RSA). From an implementation point of view a shorter key length means a higher processing speed, smaller power consumption, and silicon area requirement. Scalar multiplication is the main operation in Elliptic Curve Diffie-Hellman (ECDH), which is a key-agreement protocol using ECC. As shown in the prior literature, this operation is both vulnerable to Power Analysis attack and requires a large amount of time. Therefore, a lot of research has focused on enhancing the performance and security of scalar multiplication. In this work, we describe three schemes to counter power analysis cryptographic attacks. The first scheme provides improved security at the expense of a very small cost of additional hardware overhead; its basic idea is to randomize independent field operations in order to have multiple power consumption traces for each point operation. In the second scheme, we introduce an atomic block that consists of addition, multiplication and addition [A-M-A]. This technique provides a very good scalar multiplication protection but with increased computation cost. The third scheme provides both security and speed by adopting the second technique and enhancing the instruction-level parallelism at the atomic level. As a result, the last scheme also provides a reduction in computing time. With these schemes the users can optimize the trade-off between speed, cost, and security level according to their needs and resources.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I thank **ALLAH** for blessing me with the knowledge needed to accomplish every endeavour in my life, and I ask Him to benefit others from my work.

I would like to thank:

my wife, Manahil Almuqbil, for her patience, endless love, and support.

my supervisor, Dr. Mihai SIMA, for his directions and constructive criticisms throughout this work, which provided me with precious enlightenment toward my thesis obstacles during last two year.

the committee members , **Dr. Daniela Constantinescu** and **Dr. Michael L. McGuire**, for taking time reading my thesis and providing me with their valuable feedback.

my labmate, Dr. Hamad Alrimeih, for introducing me to the field of cryptography, as well for his support on the way.

my sponsor, Qassim University, for funding me with a scholarship.

DEDICATION

To my parents, **Mohammed Almohaimeed and Sharifah Alhassan**, my source of inspiration and motivation.

To my lovely wife, **Manahil Almuqbil**.

To my beloved son, **Tariq**.

Chapter 1

Introduction

1.1 Motivations

Elliptic curve cryptography (ECC) has been playing a significant role on secure devices since it was introduced by Koblitz [15] and Miller [21] in 1980s. The great demand for ECC is created by its shorter key size that provides an equivalent security level in comparison to the longer key size with earlier public-key cryptosystems (e.g. RSA[26]). A shorter key length means a faster processing speed, lower power consumption and smaller silicon area. However, its implementation characteristics can be targeted by the attackers. Side-channel attacks, which exploit information about the system's activity such as power consumption, can be a serious threat to the overall system security as shown by Kocher in [16]. In this dissertation, we will focus on the implementation robustness against side channel attacks, and propose a number of counter measuring to these attacks.

Figure 1.1 depicts the hierarchy of ECC mathematical operations. As it is apparent in this figure, there are three levels: scalar multiplication in the top level, point operations in the middle level, and field arithmetic operations in the bottom level. Scalar multiplication relies on point doubling and point addition. Point operations depend on field arithmetic operations: modular addition, modular subtraction, modular multiplication, and modular inversion (which is the heaviest field arithmetic operation in terms of computation cost). The scalar multiplication is an attractive target of an attacker since it is the elliptic curve core operation that involves three different dependent mathematical levels [28], as shown in Figure 1.1.

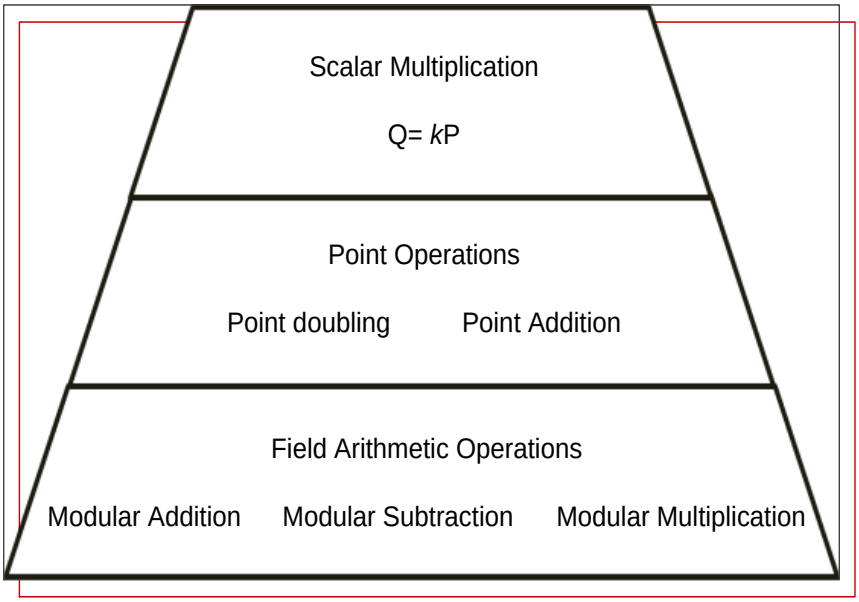

Figure 1.1: Mathematical Levels of ECC Operations

Many articles describe various approaches that enhance the robustness of the implementation against side-channel attacks and increase the computation speed at each hierarchical level of ECC. For scalar multiplication, a good review about minimizing the number of performed point operations can be found in [11]. For the middle level, different points' coordinates systems (e.g Projective and modified Jacobean coordinates) which have been introduced to minimize the computation cost of point operations can also be found in [11]. Furthermore, a number of articles introduced a new operation that is either equivalent to the point operation or a combination between them in order to reduce the computation cost as clarified in [17]. Overall, the techniques proposed in the literature to address side-channel attacks using a simple power analysis are based on:

- Inserting dummy arithmetic instructions to level the power consumption;
- Unifying both point operations formulas to level the power consumption without dummy operations;
- Using algorithms that have regular behaviour to conceal the activity inside the system.

Due to the dummy operations, the first approach can be subject to fault attacks. The second approach significantly increases the computation cost. As a result, in this work, we choose the third approach and introduce an atomic block that consist of addition, multiplication, and addition [A-M-A] for co-Z point operations. By this technique, an improvement in terms of computation cost ranging from 25% to 45% for point doubling and from 40% to 56% for point addition is achieved in comparison to prior art [6, 22, 1]. In addition, by using the proposed atomic block the computation cost is reduced about 41% comparing to the Joy's *Always Double-and-Add* algorithm that has been introduced in [25].

In terms of performance, the authors of [1], [12], and [9] enhance the level of parallelism of point operations either at the point operation level or at the field arithmetic operation level(see section 5.3), while [22], [13], and [10] introduced secure parallel schemes that enhanced the parallelism and security. In this work, we enhance the parallelism at the atomic level for point operations to provide secure and fast point operations; however, point addition is still subject to further investigation in order to be fully optimized. Our scheme provides a reduction in terms of computation time ranging from 20% to 47% in comparison to other existing work. On the other hand, different authors concentrate their effort on increasing the computing speed of field arithmetic operations; for example, [2] provides a fast modular multiplier and an arrangement of modular arithmetic computations. Our approach is different. We concentrate our effort to secure the point operations in order to secure the whole system while the computing speed is also increased over NIST recommended primes [24].

1.2 Contributions

Our main goal was to improve the robustness against side-channel attacks of point operations and increase their computing speed. The main contributions of this work can be summarized as follows:

- We propose a countermeasure that requires no penalty in term of field operations against side channel attack by randomizing the independent field arithmetic operation of each point operation.
- We propose to protect the point operations by using atomicity to secure the scalar multiplication against simple power analysis; this technique is shown to

achieve further reduction of computation cost ranging from 25% to 56% against prior art.

- We propose a protected point operation implementation that, since it has no dummy operations, has intrinsic robustness against fault attacks.
- We show a reduction in terms of computation cost for protected point operation when the co-Z representation [20] is used. We achieve up to 41% reduction in computing time in comparison to point operations that used the Joy's *Always Double-and-Add* algorithm to be protected against simple power analysis [25].
- We propose an implementation of the protected parallel point operations, where two atomic blocks can be executed in parallel.

1.3 Thesis Organization

This section provides a road map of the thesis.

Chapter 2 presents basic information on elliptic curve cryptography (ECC), where the main operations of elliptic curve are outlined: scalar multiplication with different algorithm, point operation and their representation, and field arithmetic operations.

Chapter 3 shows state-of-the-art countermeasures against power analysis attacks (both simple and differential) available in the literature.

Chapter 4 describes the architecture of our implementation. We discuss each component, in particular the accelerator, modular addition and subtraction, and modular multiplication.

Chapter 5 assesses the methodology behind the proposed techniques and their trade-offs in achieving security and computing performance.

Chapter 6 concludes the thesis and proposes research direction for future work.

Chapter 2

Introduction to Elliptic Curve

2.1 Introduction

Since Elliptic Curve Cryptography was introduced by Koblitz [15] and Miller [21] in 1980s, it has become well established due to its ability for shorter key length's to create the same security level in comparison to other public-key cryptosystems such as RSA. Table 2.1 illustrate the differences between the standard keys in term of length that have same level of security in terms of CPU cycles that are needed to crack the cryptosystem [3].

Table 2.1: NIST Guidelines for Public Key Sizes[3]

ECC Key Size (bits)	RSA Key Size (bits)	Key Size ratio
160	1024	1 : 6
256	2072	1 : 12
384	7680	1 : 20
512	15360	1 : 30

It is important to have a shorter key length size because it translates to a short processing time, less power consumption and smaller silicon area. The gap between systems grows as the key sizes increase as shown in column 3 in table 2.1. For example, ECC-160 has a six times smaller key-size than RSA-1024. Furthermore, by increasing the security level, the ECC does not slow the implementation down in comparison to RSA. Therefore, ECC could generate a few signatures while RSA generates only one. These factors accompanied with the difficulty of solving the Elliptic Curve Discrete Logarithm Problem (ECDLP) increase the demand on ECC which is applicable to

small devices such as smart cards and cell phones.

In this chapter, we briefly review the ECC algorithm and outline the most important elliptic curve operations and their specifications. Scalar multiplication $Q = kP$ is the main operation in ECC, where the point P is added $(n-1)$ times to itself, where n is length of the private key(k). Elliptic curve has three different mathematical layers: field arithmetic operations, point arithmetic operations, and scalar multiplication as we have already shown in Figure 1.1.

An Elliptic Curve E over field K is defined by a Weierstrass equation [11], as shown below:

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{2.1}$$

where: $a_1, a_2, a_3, a_4, a_6 \in K$ and and the discriminant of E, Δ, is non-zero. The condition $\triangle \neq 0$ is needed to ensure that there is only one tangent line for a given point on the elliptic curve. ECC can be defined over different finite fields K. The most important finite fields that are used to implement modern cryptosystem are the binary, prime and extension fields.

As a case study, we choose a prime field, denoted by $\mathbb{F}_P$, where p is a large prime and also represents the number of elements of the field. For a prime field $\mathbb{F}_P$, equation 2.1 simplified to equation 2.2:

$$Ep : y^2 = x^3 + ax + b \bmod p \tag{2.2}$$

where $a, b \in \mathbb{F}_P$, p is a prime number larger than 3, and the discriminant $4a^3 + 27b^2 \neq 0$.

Many algorithms have been proposed to perform scalar multiplication, such as binary and Non-Adjacent Form (NAF) methods; these two methods are outlined in the next section. Scalar Multiplication based on point addition (PA) and point doubling (PD); these two points are presented in section 2.3. Point operations rely on field arithmetic operations, such as modular addition and subtraction (MAS), modular multiplication (MM) and modular reduction (MR); these field operation are explored in section 2.4.

2.2 Scalar Multiplication Operation

Scalar multiplication $Q = kP$ is the main operation in ECC, where the point P is added $(n-1)$ times to itself, where n is length of the private key(k). Many methods of point multiplications have been proposed to enhance the performance and increase the robustness against attacks. We will start with the classic binary method, which is illustrated in the following two algorithms.

Algorithm 1 Left to Right binary method

Input: $P, k = (1, k_{l-2}, ..., k_0)$

Output: $Q = kP$

$Q \leftarrow P$
for $l = ((l-1)$ downto $0)$ **do**
$Q \leftarrow 2Q$
if $k_l = 1$ then
$Q \leftarrow P + Q$
end for
return Q

Algorithm 2 Right to Left binary method

Input: $P, k = (1, k_{l-2}, ..., k_0)$

Output: $Q = kP$

$Q \leftarrow P_\infty$
for $l = (0$ to $(l-1))$ **do**
$Q \leftarrow 2Q$
if $k_l = 1$ then
$Q \leftarrow P + Q$
end for
return Q

Algorithm [1] processes the key (k) bits from left to right, where Algorithm [2] processes it in the opposite direction. However, both algorithms perform point arithmetic operations based on the scanned bit. Point doubling followed by point addition are performed when the processed bit is 1 and only point doubling is performed if it is 0. The average number of non-zero digits in $l - bit$ key is $l/2$; therefore, the

estimated running time (the cost) of scalar multiplication (kP) is $l/2$ point additions and $(l-1)$ point doubling operations, denoted by:

$$Cost(kP) = (l-1)\ CostPD + (l/2)\ CostPA \tag{2.3}$$

Algorithm 3, called *Always Double-and-Add*, can also be used to calculate the scalar multiplication. This algorithm performs point doubling and point addition at every scanned bit of the key in order to prevent simple power attack; however, it is subject to fault attack since it performs some dummy point operations [11].

Algorithm 3 Always double-and add method

Input: $P, k = (1, k_{l-2}, ..., k_0)$

Output: $Q = kP$

```
Q ← P
for l = ((l − 1) downto 0) do
    Q ← 2Q
    Q ← P + Q
end for
return Q
```

As it is apparent in algorithm 3, the average cost of the *Always Double-and-Add* method is given by:

$$Cost(kP) = (l-1)\ Cost(PD) + (l-1)\ Cost(PA) \tag{2.4}$$

Equation 2.4 indicates that the *Always Double-and-Add* algorithm is not efficient in term of the computation cost, since it performs both point operations at every bit of the scalar.

The average number of "1" bits in the key can be reduced to $l/3$ by using NAF representations, where are 0, 1, or -1 symbols are used instead 0 and 1 in the binary representation as presented in Algorithm 4. The NAF(k) representation has five properties, k is a positive integer[11]:

1. k has a unique NAF denoted NAF(k).

2. NAF(k) has the fewest nonzero digits of any signed digit representation of k.

3. The length of NAF(k) is at most $l + 1$, where l is the length of the binary representation of k.

4. If the length of NAF(k) is l, then $2^l/3<k<2^{l+1}/3$.

5. The average density of nonzero digits among all NAFs of length l is approximately $l/3$.

Algorithm 4 Binary NAF method[11]

Input: $P, k = (1, k_{l-2}, ..., k_0)$

Output: $Q = kP$

Compute $NAF(k) = \sum_{i=0}^{l-1} k_i 2^i$
$Q \leftarrow \infty$
for $l = ((l-1)$ downto $0)$ **do**
  $Q \leftarrow 2Q$
  if $k_l = 1$ then $Q \leftarrow P + Q$
  if $k_l = -1$ then $Q \leftarrow Q - P$
end for
return Q

Algorithm 4 shows how the binary NAF method works. After computing NAF(k), the type point operation depends on the value of the scanned digit. Point doubling (PD) followed by addition (PA) is performed if the digit is 1. Point subtraction (PS) is preformed instead of PA if the scanned digit is -1. Only PD will be performed if the scanned digit is zero. By analyzing the NAFs properties, it is apparent that the expected running time (the cost) of scalar multiplication (kP) is $l/3$ point additions and $(l-1)$ point doubling operations, denoted:

$$Cost(kP) = (l-1)\ Cost(PD) + (l/3)\ Cost(PA) \tag{2.5}$$

Later, the binary NAF method was generalized to become the window NAF_w method. After that sliding window method was introduced in order to reduce the require of precomputation. A good review of NAF methods explored in [11]. However, we use in this case study classic binary algorithm to proof the concept of proposed countermeasures.

In Algorithms 1,2,3, and 4, it is apparent that the scalar multiplication based on repeating point doubling (PD) and point addition (PA) which are described in the following section.

2.3 Point Arithmetic Operations

Point doubling (PD) and point addition (PA) are the main operations of the scalar multiplier. Scalar Multiplication is defined as a sequence of $PD = 2P$ and $PA = P + Q$ operations where both P and Q are points on the elliptic curve E. In the following, the PD and PA operations can be computed based on what is called the Group Law and a natural representation that is known as affine coordinates.

For best understanding, we review the geometrical addition rule for PA, as shown in figure 2.1. Let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ be two different points on the elliptic curve E. In order to find the sum of these points,a line passing through points P and Q is first drawn. This line will intersect the elliptic curve at new point R', which is the reflection of the sum point $R = (x_3, y_3)$ of the points P and Q[11].

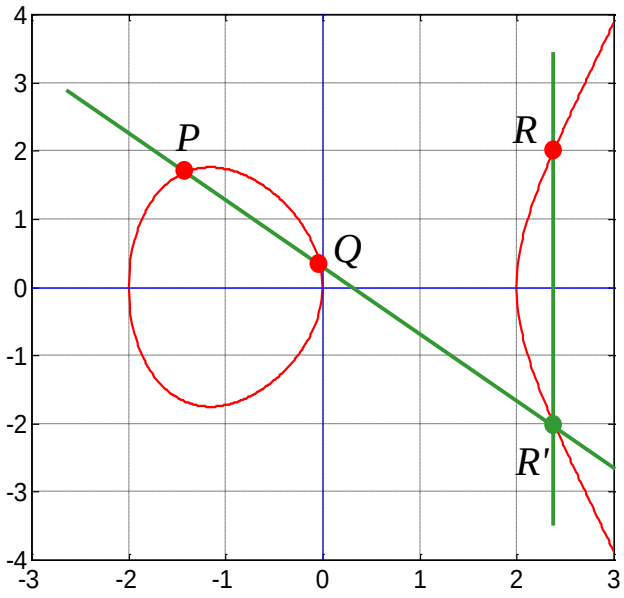

Figure 2.1: Point Addition

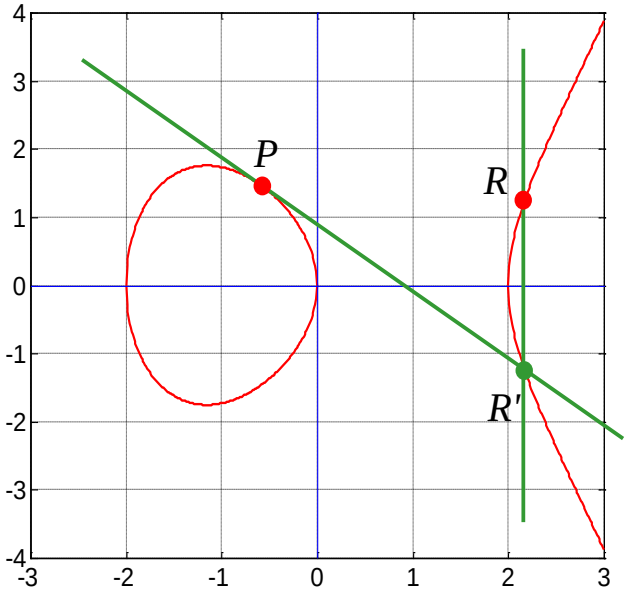

Figure 2.2: Point Doubling

PD is a special case of PA where the coordinates of both point arguments are identical $P_1 = P_2$. To compute PD, a tangent line at point P on the elliptic curve is first drawn. This tangent line will intersect the curve at new point R', which is the reflection of the point $R = 2P$, as seen in Figure 2.2 [11].

Algebraic formulas for the group law simplified from weierstrass equations are as follows[11]:

Point Addition: Let $P = (x_1, y_1)$ and $Q = (x_2, y_2) \in E(K)$, where $P \neq Q$. Then $P + Q = (x_3, y_3)$, is defined as:

$$x_3 = (\frac{y_2 - y_1}{x_2 - x_1})^2 - x_1 - x_2 \tag{2.6}$$

$$y_3 = (\frac{y_2 - y_1}{x_2 - x_1})(x_1 - x_3) - y_1 \tag{2.7}$$

The cost of point addition (PA) according to this formula is 1I + 3M.

Point Doubling: Let $P = (x_1, y_1) \in E(K)$, where $P \neq -P$. Then $2P = (x_3, y_3)$, is defined as:

$$x_3 = (\frac{3x_1^2 + a}{2y_1})^2 - 2x_1 \tag{2.8}$$

$$y_3 = (\frac{3x_1^2 + a}{2y_1})(x_1 - x_3) - y_1 \tag{2.9}$$

It is apparent that the cost of point doubling (PD) is 1I + 4M.

Many point representations have been proposed in order to reduce the computation cost by avoiding inversion I that occurs in the natural affine coordinate system [11]. In addition, it has been tried to decrease other costly field arithmetic operations, such as multiplication M and squaring S. In the following, we briefly explore the most important point representations[11]:

- **Point Doubling in Jacobian Coordinates** Let $P = (X_1, Y_1, Z_1)$ point in elliptic curve$E$. To compute $2P = (X(2P), Y(2P), Z(2P))$, the following computation is to be performed:

  $X(2P) = (3X_1^2 + aZ_1^4)^2 - 8X_1Y_1^2$,

  $Y(2P) = (3X_1^2 + aZ_1^4)((4X_1Y_1^2) - X(2P)) - 8Y_1^4$,

  $Z(2P) = 2Y_1Z_1$,

  The cost of doubling operation is $4M + 6S$.

- **Point Addition in Jacobian Coordinates** Let $P = (X_1, Y_1, Z_1)$ and $Q = (X_2, Y_2, Z_2)$ points on elliptic curve E where $P \neq \pm Q$. Hence, the point addition is computed by:

  $X_3 = (Z_1^3Y_2 - Z_2^3Y_1)^2 - (Z_1^2X_2 - Z_2^2X_1)^3 - 2Z_2^2X_1(Z_1^2X_2 - Z_2^2X_1)^2$,

  $Y_3 = (Z_1^3Y_2 - Z_2^3Y_1)(Z_2^2X_1(Z_1^2X_2 - Z_2^2X_1)^2 - X_3) - Z_2^3Y_1(Z_1^2X_2 - Z_2^2X_1)^3$,

  $Z_3 = Z_1Z_2(Z_1^2X_2 - Z_2^2X_1)$,

  Thus, the cost of point Addition in Jacobian is $12M + 4S$.

- **Mixed Addition in Jacobian-Affine Coordinates** Let $P = (X_1, Y_1, Z_1)$ and $Q = (X_2, Y_2)$ points represented by Jacobian and Affine respectively. Both points on elliptic curve E. The point addition is calculated as follows:

  $X_3 = (Z_1^3Y_2 - Y_1)^2 - (Z_1^2X_2 - X_1)^3 - 2X_1(Z_1^2X_2 - X_1)^2$,

  $Y_3 = (Z_1^3Y_2 - Y_1)(X_1(Z_1^2X_2 - X_1)^2 - X_3) - Y_1(Z_1^2X_2 - X_1)^3$,

$Z_3 = Z_1(Z_1^2 X_2 - X_1)$,

With the above equations, the cost of mixed addition is $8M + 3S$.

- **Point Doubling in Co-Z Coordinates** Let $P = (X_1, Y_1, Z_1)$ be a point in Jacobian coordinates. By setting $Z_1 = 1$, the cost drops to only $1M + 5S$ by following these formulas:

  $X(2P) = (3(X_1^2 - a)^2 - 4((X_1 + Y_1^2)^2 - X_1^2 - Y_1^4$,

  $Y(2P) = (3(X_1^2 - a)((2(X_1 + Y_1^2)^2 - X_1^2 - Y_1^4) - X(2P)) - 8Y_1^4$,

  $Z(2P) = 2Y_1$.

  Figure 2.3 show the corresponding flowchart to the above formulas. In this flowchart, a few modular addition, subtraction, squaring, and multiplication performs point doubling which more efficient in comparison to the previous representations.

- **Point Addition in Co-Z Coordinates** In[20], Meloni introduced the co-Z arithmetic that is new point addition formula where two **Jacobian** points share the same z-coordinate. Let $P = (X_1, Y_1, Z)$ and $Q = (X_2, Y_2, Z)$ points on elliptic curve E where $P \neq \pm Q$. So adding two points is performed as follows:

  $X_3 = (Y_1 - Y_2)^2 - X_1(X_1 - X_2)^2 - X_2(X_1 - X_2)^2$,

  $Y_3 = (Y_1 - Y_2)(X_1(X_1 - X_2)^2 - X_3) - Y_1(X_1(X_1 - X_2)^2 - X_2(X_1 - X_2)^2)$,

  $Z_3 = Z(X_1 - X_2)$,

  Figure 2.4 shows the flowchart of point addition where number of field arithmetic operations are involved. This operation cost only $5M + 2S$ operations. This operation is more efficient than in all the other representation.

Due to its lower cost, we decide to use and secure the co-z representation. We implement EC point operations based on co-z coordinate over NIST primes which are proven to be secure. NIST recommends five primes p which their coefficients a is -3 and b satisfying $b^2 c \equiv -27 \pmod{p}$, the coordinates of the base point P (x and y), the order n of the base point P, and the curve cofactor his the same for all five prime

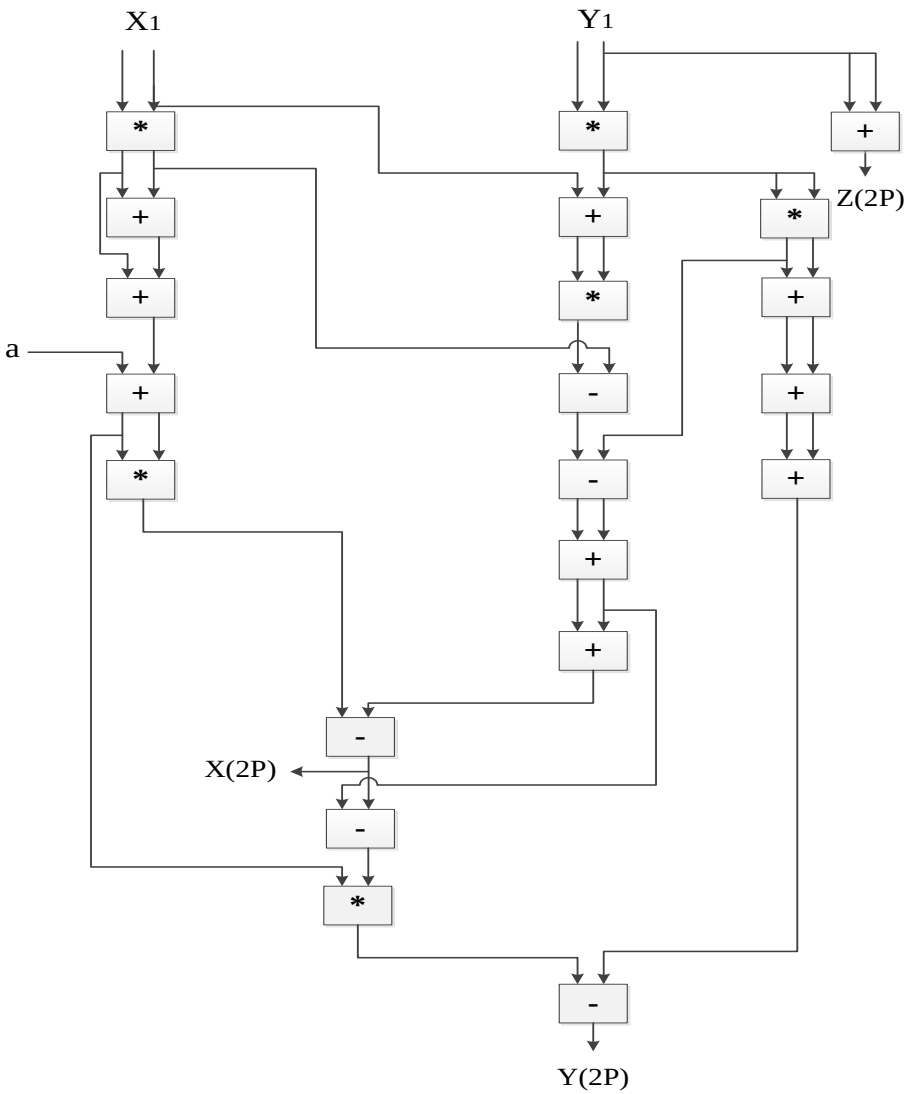

Figure 2.3: Point Doubling in Co-z

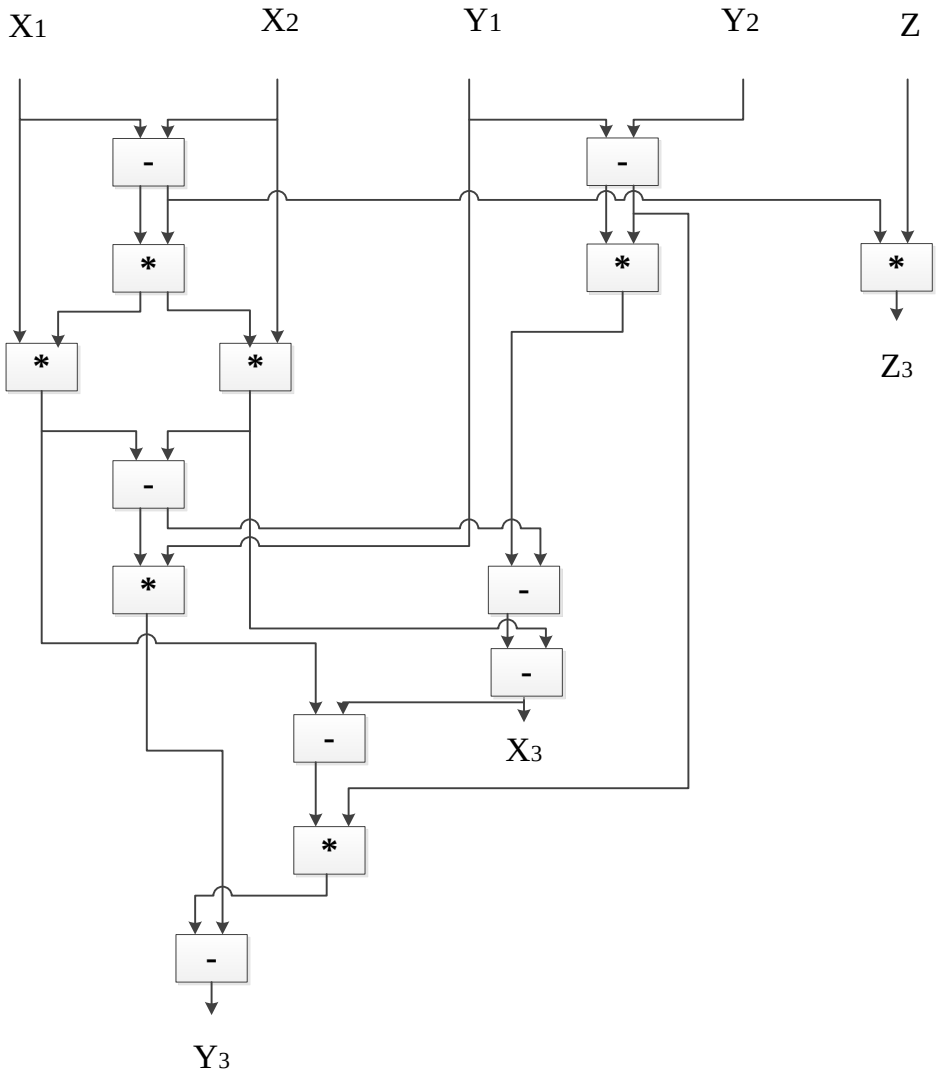

Figure 2.4: Point Doubling in Co-z

Table 2.2: Parameters for p-192 of NIST primes

P-192:
$p = 2^{192} - 2^{64} - 1$, $h = 1$, $a = -3$
b = 0x 64210519 E59C80E7 0FA7E9AB 72243049 FEB8DEEC C146B9B1
n = 0x FFFFFFFF FFFFFFFF FFFFFFFF 99DEF836 146BC9B1 B4D22831
x = 0x 188DA80E B03090F6 7CBF20EB 43A18800 F4FF0AFD 82FF1012
y = 0x 07192B95 FFC8DA78 631011ED 6B24CDD5 73F977A1 1E794811

($h = 1$) as shown in Table 2.3 [24].

Table 2.3 shows the parameter of prime p-192 that we use in our implementation.

2.4 Field Arithmetic Operations

Both point doubling (PD) and point addition (PA) operations require a different number of modular addition and subtraction (MAS), and modular multiplication (MM) operation, where each has its own properties. In the next section, we illustrate their function and algorithms.

2.4.1 Modular Addition and Subtraction

In modular addition,two elements x and y $\in [0, p-1]$ are added:$(R = (x+y) \bmod p)$. If $R \geq p$, then p is subtracted from R. Similarly, in modular subtraction, two elements x and y $\in [0, p-1]$ are subtracted:$(R = (x-y) \bmod p)$. If $R<0$, then p is added to R. For instance, $(15 + 20) \bmod 21 = (35 - p) = 14$. Modular addition and subtraction are combined to be performed in one algorithm as shown in Algorithm5. In term of hardware cost, we consider that addition is equal to subtraction ($A = S$) because both are performed by a single component in hardware.

Algorithm 5 MAS[28]

Input: $x, y \in [0, p-1], AS \in 0, 1$, prime p

Output: $R = (x + y) \bmod p$ if $AS = 0$, else $R = (x - y) \bmod p$

IF $AS = 0$ then $R \leftarrow x + y$
IF $R \geq p$ then $R \leftarrow R - p$
Else if $AS = 1$ then $R \leftarrow x - y$
IF $R<0$ then $R \leftarrow R + p$
return R

Moreover, the most important properties for MAS used in algorithm 5 are:

- It is commutative, where $(x + y) \bmod p = (y + x) \bmod m$.
- It is associative, where $((x + y) + z) \bmod p = (x + (y + z)) \bmod p$.
- It has a natural element (0), where $(x + 0) \bmod p = x \bmod p$.

2.4.2 Modular Multiplication

In modular multiplication, two elements x and y $\in [0, p-1]$ are multiplied:$(R = (x \times y) \bmod p)$. Multiplication is the second most expensive modular operation in term of execution time and consumed power. Therefore, many methods have been proposed to enhance the performance of modular multiplication, for example, standard multiplication, interleaved method, and Montgomery approach [28].

Algorithm 6 Standard Multiplication[28]

Input: $l - bitx, y \in [0, p-1]$, prime p

Output: $R = (x \times y) \bmod p$

$P \leftarrow 0$
for $i = (0$ to $(l-1))$ **do**
  $P \leftarrow (2P + x_{l-1-i}.y)$
  $R = Pmodp$
end for
return R

Algorithm 6 shows how standard modular multiplication is performed using shift and add operations that applied to l-bit inputs x and y. First, state 2 generates

$2l$ result P by computing partial products $(x_{l-i-1}.y)$ and left shifts the previous intermediate result $(2P)$, and adds the new intermediate result. Finally, the last state performs reduction to ensure $R \in [0, p-1]$. To proof the concept of our contributions, we choose the standard multiplication algorithm shown in Algorithm 6.

Algorithm 7 Interleaved Modular Multiplication[28]

Input: $l - bit x, y \in [0, p-1]$, prime p
Output: $R = (x \times y) \bmod p$

```
P ← 0
for i = (0 to (l − 1)) do
    P ← (2P + x_{l−1−i}.y)
    R := P mod p
end for
return R
```

Likewise, Algorithm 7 performs modular multiplication; however, to satisfy $R \in [0, p-1]$, two subtraction operations are performed at most as follows:

$$P' := P - n; \quad \text{If } P' \geq 0 \quad then \quad P = P'$$
$$P' := P - n; \quad \text{If } P' \geq 0 \quad then \quad P = P'$$

In[23], P. L. Montgomery presented Montgomery modular multiplication as alternative method in 1985. According to this method, two integers are multiplied modulo p, where these integers are beforehand converted from the standard representation to a Montgomery representation. In order to transform a number x into the Montgomery domain, we need to compute $x * Rmod(p)$; where R is the smaller power of the base that is greater than the modulus. The main feature of Montgomery method is replacing division with less expensive operations during reduction, as shown in Algorithm 8 and 9.

Algorithm 8 Montgomery Product[28]

Input: $x_{Mon}, y_{Mon}, r = 2^l$, an prime p, pre-computed p'.

Output: $R = MonPro(x_{Mon}, y_{Mon}) = (x_{Mon} \cdot y_{Mon} \cdot r^{-1}) \bmod p$

$u \leftarrow x_{Mon} \cdot y_{Mon}$
$v \leftarrow u \cdot p' \bmod r$
$R \leftarrow (u + v \cdot p)/r$
$If R \geq p$ then
return $R - p$
$Else$
return R

Algorithm 9 Montgomery Modular Multiplication[28]

Input: $x, y, r = 2^l$, an prime p.

Output: $R = x \cdot y \bmod p$

$x_{Mon} \leftarrow x \cdot r \bmod p$
$y_{Mon} \leftarrow y \cdot r \bmod p$
$R_{Mon} \leftarrow MonPro(x_{Mon}, y_{Mon})$
$R \leftarrow MonPro(R_{Mon}, 1)$
return R

2.4.3 Modular Reduction

Modular reduction is the last step of standard modular multiplication that reduces the length of result R from $2l$ to l. In general, a number of approaches have been proposed to ensure $R \in [0, p-1]$. For example, Restoring and Non-restoring Division Algorithm, and Barrett Reduction Algorithm [27] [4]. The performance of elliptic curve schemes depends heavily on the speed of field multiplication; therefore, it is a good reason to use selected moduli along with standard modular multiplication in our thesis, such as the NIST-recommended five. These primes permit fast reduction as shown in the following algorithms[11][24]:

$p_{192} = 2^{192} - 2^{64} - 1$
$p_{224} = 2^{224} - 2^{96} + 1$
$p_{256} = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$
$p_{384} = 2^{384} - 2^{128} - 2^{96} + 2^{32} - 1$
$p_{521} = 2^{521} - 1.$

Algorithm 10 Fast reduction modulo $p_{192} = 2^{192} - 2^{64} - 1$

Input: An integer $c = (c5, c4, c3, c2, c1, c0)$ in base 2^{64} with $0 \leq c < p_{192}^2$

Output: $cmodp_{192}$

Define $192 - bit$ integers:
s1 = (c2, c1, c0);
s2 = (0, c3, c3);
s3 = (c4, c4,0);
s4 = (c5, c5, c5).
return $(s1 + s2 + s3 + s4 \bmod p_{192})$

Algorithm 11 Fast reduction modulo $p_{224} = 2^{224} - 2^{96} + 1$

Input: An integer $c = (c13, ..., c2, c1, c0)$ in base 2^{32} with $0 \leq c < p_{224}^2$

Output: $cmodp_{224}$

Define $224 - bit$ integers:
s1 = (c6, c5, c4, c3, c2, c1, c0);
s2 = (c10, c9, c8, c7,0,0,0);
s3 = (0, c13, c12, c11,0,0,0);
s4 = (c13, c12, c11, c10, c9, c8, c7);
s5 = (0,0,0,0, c13, c12, c11).
return $(s1 + s2 + s3 - s4 - s5 \bmod p_{224})$

Algorithm 12 Fast reduction modulo $p_{256} = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$

Input: An integer $c = (c13, ..., c2, c1, c0)$ in base 2^{32} with $0 \leq c < p_{256}^2$

Output: $cmodp_{256}$

Define $256 - bit$ integers:

s1 = (c7, c6, c5, c4, c3, c2, c1, c0);

s2 = (c15, c14, c13, c12, c11,0,0,0);

s3 = (0, c15, c14, c13, c12,0,0,0);

s4 = (c15, c14,0,0,0, c10, c9, c8);

s5 = (c8, c13, c15, c14, c13, c11, c10, c9);

s6 = (c10, c8,0,0,0, c13, c12, c11);

s7 = (c11, c9,0,0, c15, c14, c13, c12);

s8 = (c12,0, c10, c9, c8, c15, c14, c13);

s9 = (c13,0, c11, c10, c9,0, c15, c14).

return $(s1 + 2s2 + 2s3 + s4 + s5 - s6 - s7 - s8 - s9 \bmod p_{256})$

In our thesis, we have chosen Prime p_{192} to gain fast reduction since the main goal is enhancing the level of parallelism and resist the side channel attacks (SCAs). This will be described in Chapter 3.

Chapter 3

Side Channel Attacks and Countermeasures

3.1 Introduction

Cryptographic devices (e.g. smart cards, mobile phone, and RFID tags) play a major role in the modern society. These devices are threatened by two different types of attacks: *Active attacks* and *Passive attacks*[19]. Revealing the secret key is the main goal of these attacks. First, active attacks are manipulating the cryptographic device's inputs and environment in order to induce abnormal behaviour in the device under attack. Second, passive attacks are extracting the secret key by monitoring the physical properties of the cryptographic device such as SCA[19]. In this report, we focus on both types of Side Channel Attacks (SCAs): Simple Power Analysis (*SPA*) and Differential Power Analysis (*DPA*) attacks.

3.2 Simple Power Analysis

Simple Power Analysis (SPA) and Differential Power Analysis (DPA) were both introduced by Kosher et al [16] in 1998. SPA is revealing the key by monitoring a single or view power traces. These traces provide sufficient information to exploit the secret information of algorithms because these algorithms have conditional branches where their operations rely on the scanned bit. To illustrate that, lets take Elliptic Curve digital Signature Algorithm (ECDSA) from Elliptic Curve Cryptography (ECC) as an example. The elliptic curve point result of the scalar multiplication is essential in

this algorithm. The scalar multiplication is considered as a series of point doubling (PD) and addition (PA) operations that depend on the scanned bit of the key. Point doubling is performed when the bit is 0. Also, point doubling is followed by point addition if the scalar bit is 1. Many countermeasures have been proposed to resist the simple power analysis, as discussed in the following section.

3.2.1 Countermeasures against Simple Power Analysis

In simple power analysis, the key can be exposed by monitoring EC point operations such as point doubling (PD) and point addition (PA), as mentioned above. The number of the field arithmetic operations on PD and PA are different; therefore, the PD can be simply distinguishable from PA in term of power consumption traces, as it is apparent in Figure 3.1.

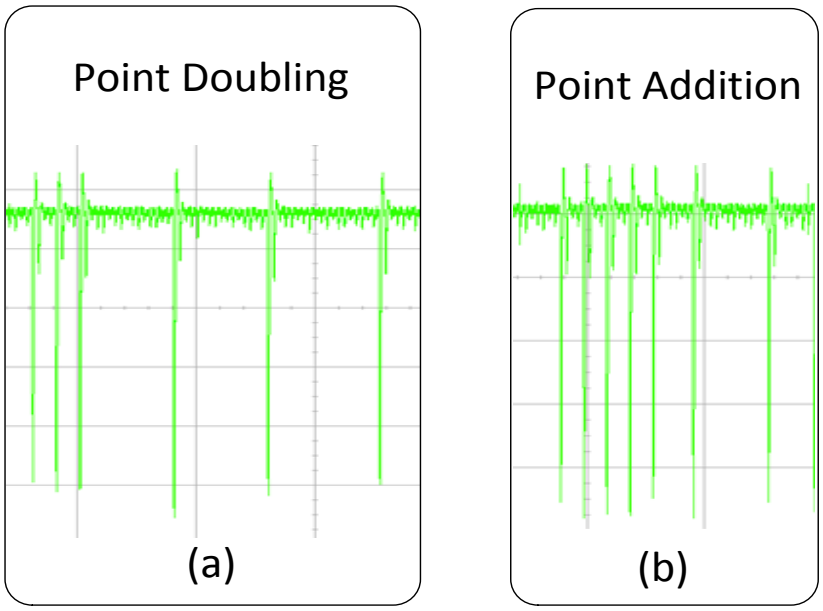

Figure 3.1: Power consumption Traces of Point Operation

Figure 3.1 shows that the traces of point operation can be distinguished. Figure 3.1 (a) shows the trace of point doubling and (b) show the trace of point addition; it is apparent that they look differently. So, if the pattern (a) is repeated, that means the scanned bit of the key is "0". Also, if the pattern (a) is followed by the pattern (b), that means the scanned bit of the key is "1".

In order to eliminate the weakness against SPA, it is important to eliminate any link between the observed information and the scanned bit i. In [7] [14], several approaches to obtain this purpose have been presented:

- Inserting dummy arithmetic instructions to level the power consumption;
- Unifying both point operations formulas to level the power consumption without dummy operation;
- Using algorithms that have regular behaviour to conceal the activity inside the system.

These techniques are explained below.

Using algorithms that have regular behavior:

Using regular point multiplication algorithms allows the PD and PA to have different power traces. Furthermore, they do not lead to the scanned bit of the key because there is no conditional branch in the algorithm. For instance, *Always Double-and-Add* algorithm performs point doubling (PD) followed by point addition (PA) for each bit of the key no matter what its value as it is shown in Algorithm below [14].

Algorithm 13 Always Double-and-Add Algorithm

Input: $P, k = (1, k_{l-2}, ..., k_0)$

Output: $Q = kP$

```
Q ← P
for l = (k − 2 downto 0) do
    Q_0 ← 2Q
    Q_1 ← P + Q_0
    Q ← Q_{k_l}
end for
return Q
```

In [6], Chevallier-Mames, Ciet and Joye generalized the idea behind the *Always Double-and-Add* algorithm by introducing the side-channel atomicity. Point doubling and point addition are represented by a multiple of atomic blocks that have each the same set of modular operations. This technique leads to further reduction in term of

computation cost.

In this dissertation, we propose an atomic block that consist of addition, multiplication, and addition [A-M-A]. We consider that the addition and subtraction are alike from a power consumption point of view since they are performed via same component in hardware. As a case study, we applied this atomic block on crypto algorithm using Co-Z point operations[25].

Another example of using regular point multiplication is using Montgomery Point Multiplication Algorithm 14 which seems to be similar to *Always Double-and-Add* at the first look. However, point doubling (PD) and point addition (PA) will be performed at every scanned bits zeros or non-zeros with no dummy operations. The k_l is the value of the scanned scalar bit at every iteration. Not only that, it computes the sum of two points without y-coordinate. Therefore, this algorithm is more cost-effective.

Algorithm 14 Montgomery Point Multiplication Algorithm [2]

Input: $P, k = (1, k_{l-2}, ..., k_0)$
Output: $Q = kP$

$Q_0 \leftarrow P$
$Q_1 \leftarrow 2P$
for $l = (k-2)$ downto 0 **do**
  $Q_{1-k_l} \leftarrow Q_0 + Q_1$
  $Q_i \leftarrow 2Q_{k_l}$
end for
return Q_0

Inserting dummy field arithmetic operation:

According to this technique, the power consumption of point operation is leveled by inserting dummy field arithmetic operations. This approach is simple to implement; however, it has drawbacks such as increasing the computation cost. As a result of these extra dummy operations are involved, the implementation might be subject to fault attack.

As we can see from table 3.1 that point doubling has one dummy operation in order to be balanced with point addition. As result, both operation has same power

Table 3.1: Point Operations (with dummy operation)

Round	Point Doubling ($P_1 = P_1 + P_2$)	Point Addition ($P_1 = 2P_1$)
1	$T_0 = x_0 + x_1$	$T_3 = x_0 + x_1$ (Dummy)
2	$T_1 = y_0 + y_1$	$T_3 = x_1 + T_3$
3	$T_2 = T_1/T_0$	$T_2 = y_0/x_0$
4	$T_0 = T_0 + T_2$	$T_2 = x_0 + T_2$
5	$T_3 = T_2^2$	$T_0 = T_2^2$
6	$T_3 = T_3 + a_2$	$T_0 = T_0 + a_2$
7	$T_0 = T_0 + T_5$	$T_0 = T_0 + T_2$
8	$T_1 = T_0 + y_1$	$T_1 = T_0 + T_1$
9	$T_3 = T_0 + x_1$	$T_3 = T_0 + T_3$
10	$T_2 = T_2 \times T_3$	$T_2 = T_2 \times T_3$
11	$T_1 = T_1 + T_2$	$T_1 = T_1 + T_2$

consumption pattern. Therefore, using this technique increase the robustness against simple power analysis.

Unifying both point operations formulas:

Unifying both point operations formulas allows a set of identical modular operations to be performed independent on the type of operations occurred. In [5], Brier and Joye noticed that they can unify the slope of point operations in elliptic curve given by Weierstrass form:

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{3.1}$$

Let $P_1 \neq P_2$ and both points on the curve, therefore the sum of these points $P_3(x_3, y_3) = P_1 + P_2$ is defined as follows:

$$x_3 = \lambda^2 + a_1\lambda - a_2 - x_1 - x_2, \qquad y_3 = \lambda(x_1 - x_3) - y_1 - a_1x_3 - a_3 \tag{3.2}$$

where:

$$\lambda = \begin{cases} \dfrac{y_1 - y_2}{x_1 - x_2} & \text{if } x_1 \neq x_2, \quad (3.3) \\ \dfrac{3x_1^2 + 2a_2x_1 + a_4 - a_1y_1}{2y_1 + a_1x_1 + a_3} & \text{if } x_1 = x_2, \quad (3.4) \end{cases}$$

The above slope λ equations can be unified for both point addition and doubling by rewriting equation 1.3 and 1.4 with 1.1 as end by:

$$\lambda = \frac{x_1^2 + x_1x_2 + x_2^2 + a_2x_1 + a_2x_2 + a_4 - a_1y_1}{y_1 + y_2 + a_1x_2 + a_3} \tag{3.5}$$

The main advantage of this technique is unifying the power consumption of point operations with no extra dummy operation. As a result, no extra dummy operation are involved, the implementation intrinsically robust against fault attacks.

3.3 Differential Power Analysis

Differential power analysis (DPA) along with simple power analysis (SPA) were introduced by Kosher et al [16] in 1998. DPA is an extended version to SPA where the attackers need more power traces to be able to eliminate the noise in order to get valuable information from the power traces. However, in differential power analysis (DPA) no knowledge of the cryptographic device implementation is needed as long as the cryptographic algorithm is well known[19].

3.3.1 Countermeasures against Differential Power Analysis

In [8], Coron described in more detailed three different countermeasures that based on introducing random number while computing point multiplication $Q = kP$:

Randomizing the Private Exponent

We can randomize the private exponent to counter DPA by adding multiple of $\#E$. Now,$k' = k + r\#E$ where r is selected random number. Since $(r\#E)P$ is equal O, $k'P$ is equal to kP. This approach changes the k' at each execution time of $Q = k'P$ which is harden the attacks.

Blinding Point P

Blinding point P could be use to resist DPA by computing the scalar multiplication $Q = k(R + P) = kR + kP$. By the end of the computation, kR is subtracted from Q to get kP since the R and $S = kR$ stored in the memory of such cryptographic devices. Furthermore, R and S computed at each execution time by $R \leftarrow (-1)^b 2R$ and $S \leftarrow (-1)^b 2S$, where b is a random bit generated at the same time.

Randomizing Projective Coordinates

To resist DPA, randomizing the projective coordinates can be done before each new execution of scalar multiplication or after each point operation. Therefore, the attacker will not be able to predict at any bit the point P in projective coordinates. This randomization is done by the following equation:

$(X, Y, Z) = (\lambda X, \lambda Y, \lambda Z)$; where $\lambda \neq 0$ in the finite field.

In this dissertation, we focus on increasing the robustness against simple power analysis (SPA). So, we use the proposed atomic block to prevent co-z point operation from SPA. Randomizing independent field arithmetic operation is another method used to resist SPA. These techniques are explained in chapter 5.

Chapter 4

Hardware Implementation of Secure Point Operations

4.1 Overall Architecture

In this chapter, we describe comprehensively our overall system architecture and how it works. We assume that we have software that takes the control over accelerator by sending two points with the scaler and the prime to perform the scalar multiplication. Then, it receives the result point.

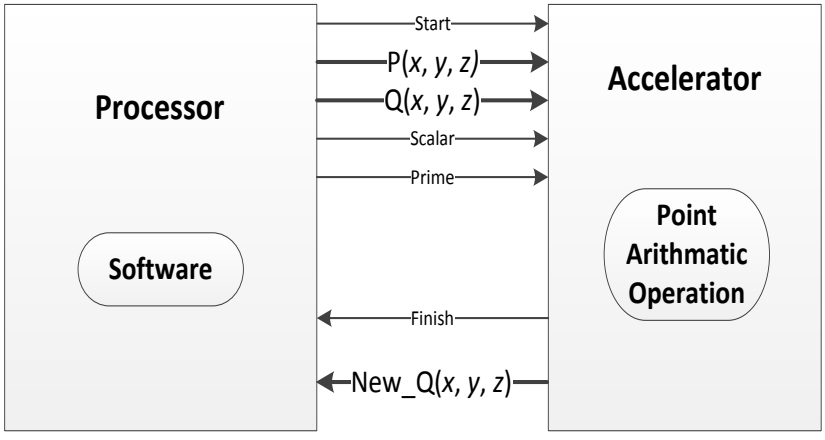

Figure 4.1: Overall System Architecture

Figure 4.1 shows the overall system architecture. In this architecture, the processor execute the software and when it reaches to scalar multiplication, it sends points to the accelerator and wait for the result. The accelerator will send back the result when it has completed its function.

For the communication between the processor and the accelerator, we used a handshaking mechanism as illustrated in 4.2. The processor sends point P and Q which is in co-z representation. Along with the two points, the accelerator receives the scalar. Then, it enables the *Start* signal and do other independent tasks entrusted to it. As soon as the accelerator receives *Start* signal, it starts performing the scalar multiplication and send back the new value of Q and informs the processor that the job has been completed by sending *Finish* signal. When the processor receives *Finish* signal, the accelerator waits until next call.

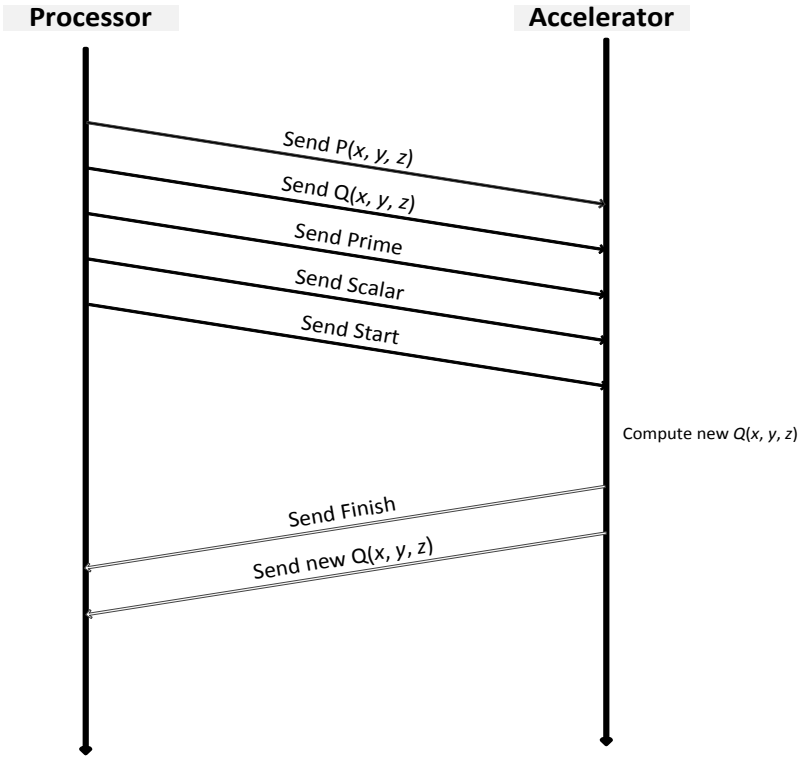

Figure 4.2: Handshaking Protocol

Figure 4.3 shows the connection between processor and accelerator which performs the scalar multiplication. The accelerator receives the two point, the scalar, and the prime that all are 192-bit. When the accelerator is done, it sends the result back with

same length.

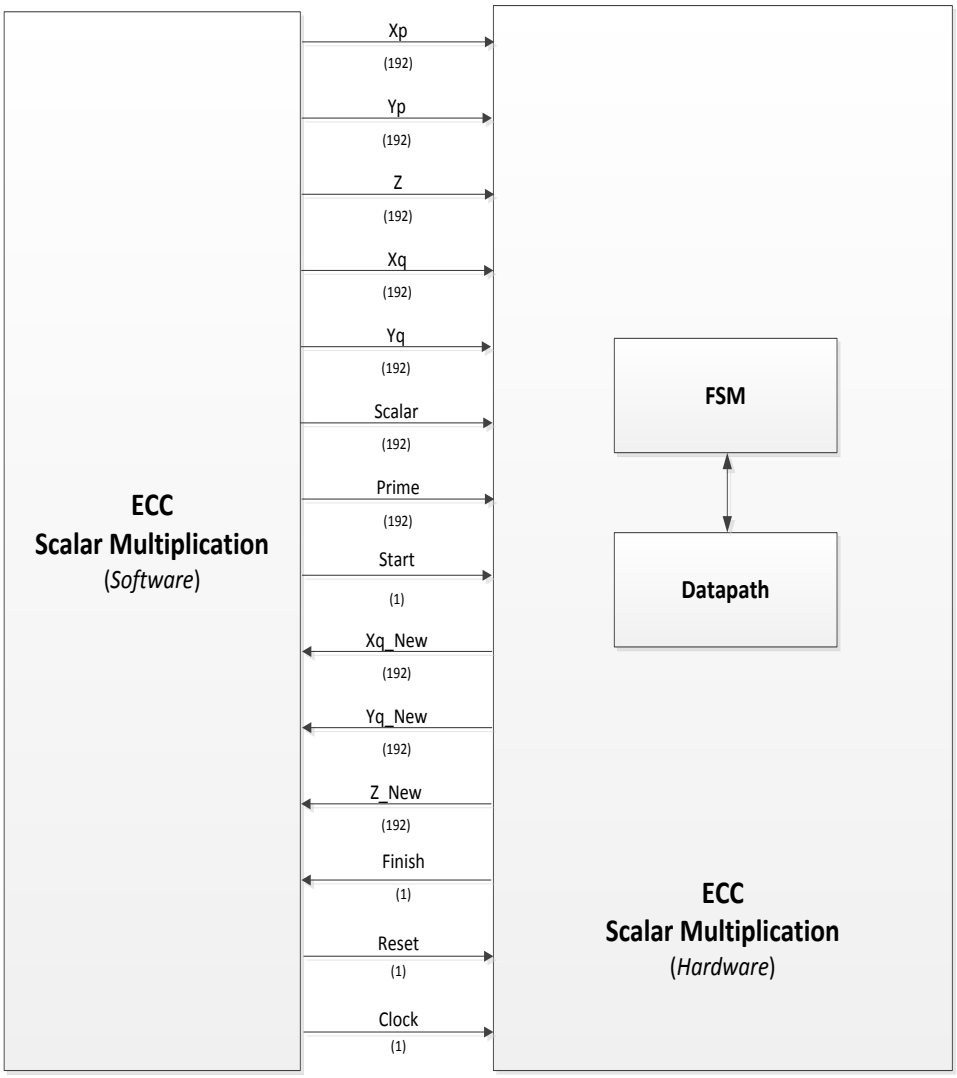

Figure 4.3: PIN Connection between processor and accelerator

4.2 Accelerator

The accelerator implemented in hardware to accelerate the computation which is our concern. The accelerator has two important modules, the Finite State Machine (FSM) and datapath as shown in 4.4. FSM part is responsible for controlling the

communication with the processor and generating control signals to datapath. The datapath is the computation part of the accelerator. It consists of one multiplier and one adder/subtracter, multiplexers and registers.

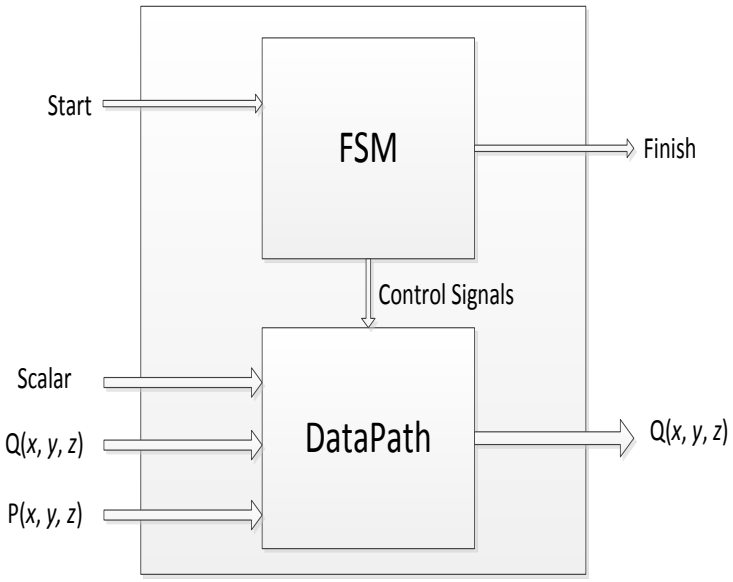

Figure 4.4: Accelerator Architecture

As shown in figures 2.4 and 2.3, the point operations could be broken down into number of modular additions and multiplications with their dependencies. Looking to these figures, point addition requires 7 multiplications and 7 additions and point doubling requires 6 multiplications and 15 additions. Since we implement the point operation in hardware, we assume multiplication is equal to squaring and addition and subtraction are alike.

Due to the limitation of our target FPGA device, we choose to implement the proposed datapath with one plain multiplier and one plain adder. In addition, we used some register to save the intermediate values to their dependant and sequentially perform the whole computation. Therefore, the FSM part has to generate several extra controlling signals to manage the data flow through multiplier, adder, and registers bank. Moreover, this implementation minimizes the hardware resources while has long latency. This implementation is not competitive in term of performance; however, our goal is to demonstrate the concepts of the proposed countermeasures..

As mentioned earlier, the processor sends two points, P and Q to the accelerator

and wait for the result point new Q. After the accelerator finishes its function, it sends *finish* signal to the processor and sends back the result. In this implementation, we assume that the processor sends points in co-z representation and gets back the result in same format.

In this implementation, we support only one prime, p-192, of NIST recommended primes for the ECC point operations[24]. Since the accelerator receives two point in co-z representation, we need five registers to capture the 192-bit inputs as shows in figure 4.5. The accelerator receives the input in subwords, 32 bits, because of the limited number of IOs pins in the Spartan-6 chip that we used in our experimental set up. Also, we need two extra registers to capture the scalar and the prime. For performing point operation, we have three modular field arithmetic operations: multiplication, addition, and subtraction as shown in figures 2.4 and 2.3. We combined the addition and subtraction in one entity and distinguish between them by AS_Type signal. AS_Type signal is either zero for addition or one for subtraction; more detailed presented in section 4.3. Moreover, multiplication is another entity which performs both multiplication and squaring; this entity explained in section 4.4. Since we have only one multiplier and one adder as seen in 4.5 , a register bank that consists of ten registers is needed to keep the intermediate values along with the result and the updated point P for their need. As result of having one multiplier and adder, we use 7 multiplexers to route the registers to the multiplier and the adder. Furthermore, we use another three 6-to-1 multiplexers to deliver the output in sub-words to the processor.

To control the whole process, an ECC finite state machine is implemented. The FSM also controls handshaking signals between the processor and the accelerator. In addition, it is responsible for managing the flow between the components inside accelerator.

In the next two section, we will explore modular field arithmetic that used in this implementation.

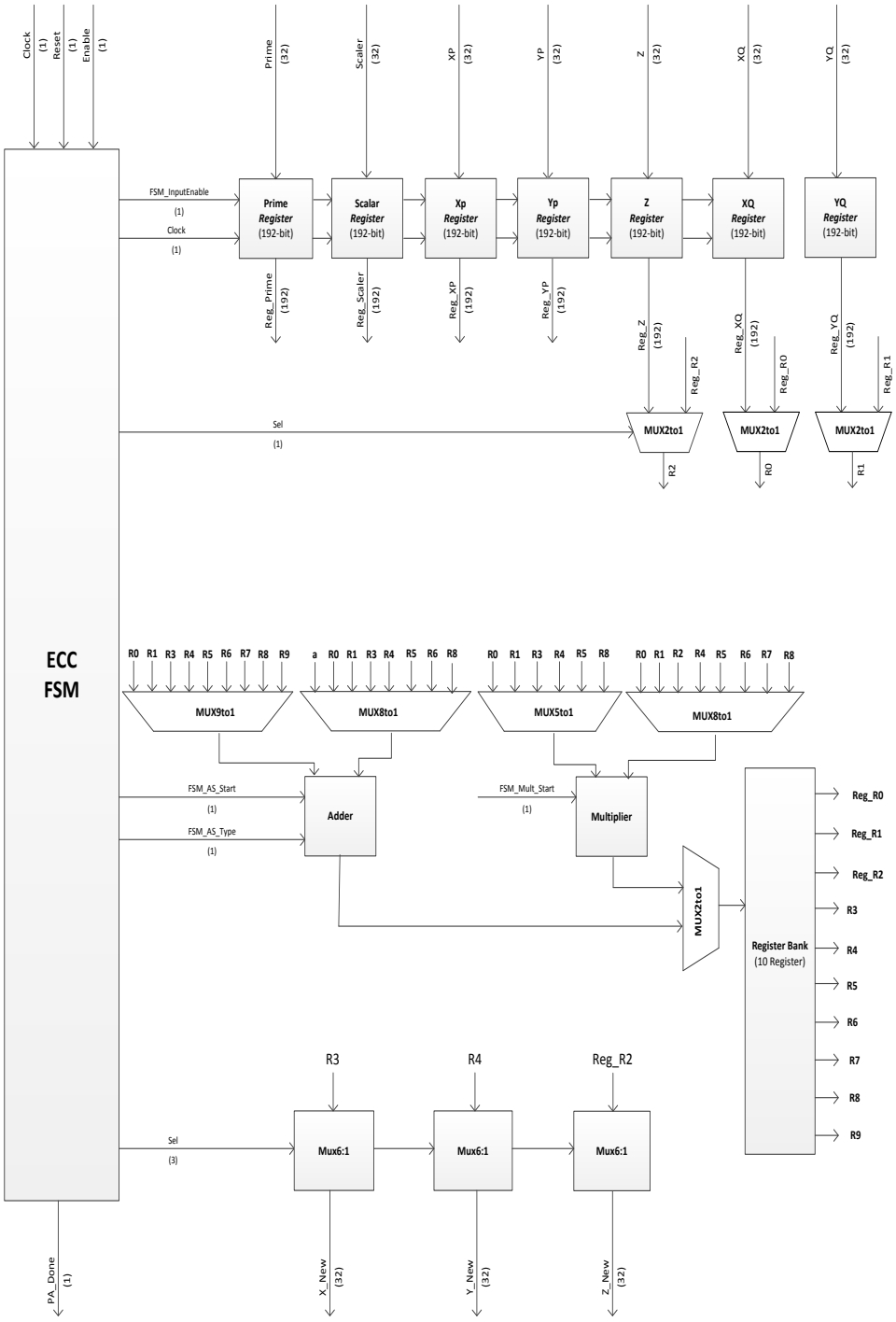

Figure 4.5: Accelerator Architecture

4.3 Modular Adder/Subtracter (MAS)

We implemented one entity that has the capability of performing addition and subtraction. The operation specified by AS_Type signal zero for addition and one for subtraction. Figure 4.6 shows the block diagram of modular adder/subtracter where it receives two 192-bit input operands from multiplexers or registers as mentioned before and the prime. MAS enabled and specified the operation by two signals from *ECC FSM*. It also informs the *ECC FSM* when it has done the operation and produce the result to the register bank.

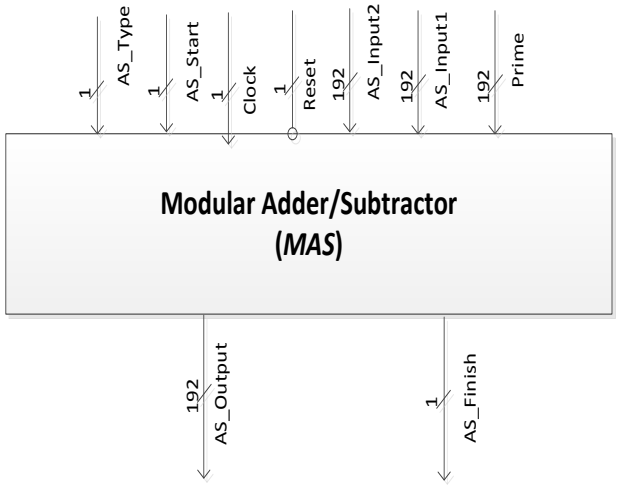

Figure 4.6: Block Diagram of Modular Adder/Subtracter

Figure 4.7 elucidates the internal logic diagram of the top level adder architecture where there are two register to keep the input operands and one register to store the output. Another register is needed to store the prime for performing reduction if needed. To control the internal signals and external signal in adder, *AddSub FSM* is needed.

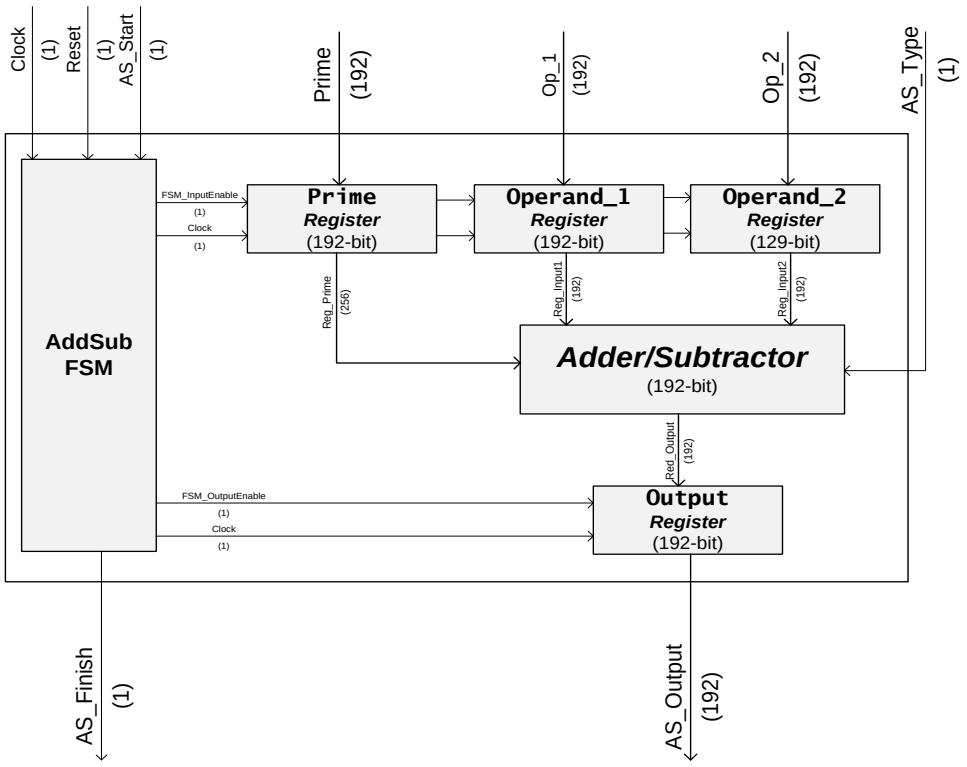

Figure 4.7: Logic Diagram of MAS

Figure 4.8 shows the behaviour of the FSM which is consists of three states *wait*, *output*, and *finish*. The adder will stay in state *wait* until it receives AS_Start signal from the *ECC FSM*. Simultaneously, *AddSub FSM* enables the inputs to be loaded in registers. In the next cycle, addition with reduction will be done and store the result in the output register. The adder remains in *finish* state until the AS_Start signal return zero. Finally, *AddSub FSM* sends AS_Finish signal one to inform the *ECC FSM* that the addition is completed.

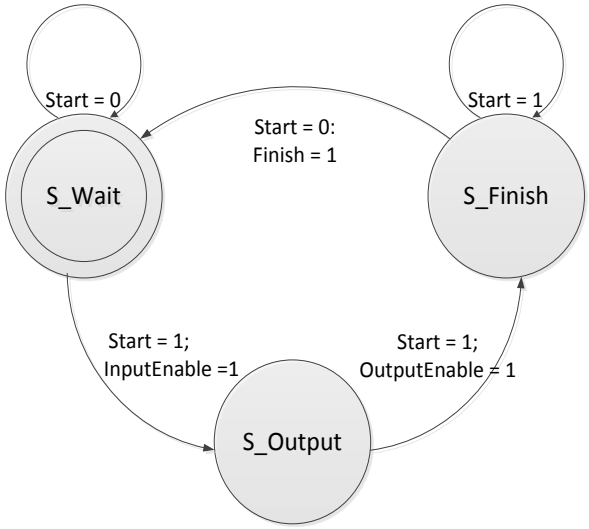

Figure 4.8: MAS Finite State Machine

4.4 Modular Multiplication and Reduction

The modular multiplication is another modular arithmetic operation that needed to perform the point doubling and point addition. Similarly, the modular multiplication is implemented where it receives two 192-bit inputs and produce the result as 192-bit out after performing reduction. Figure 4.9 represents the block diagram of the modular multiplier.

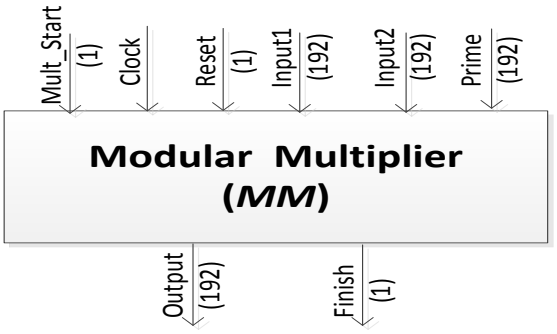

Figure 4.9: Block Diagram of Modular Multiplier

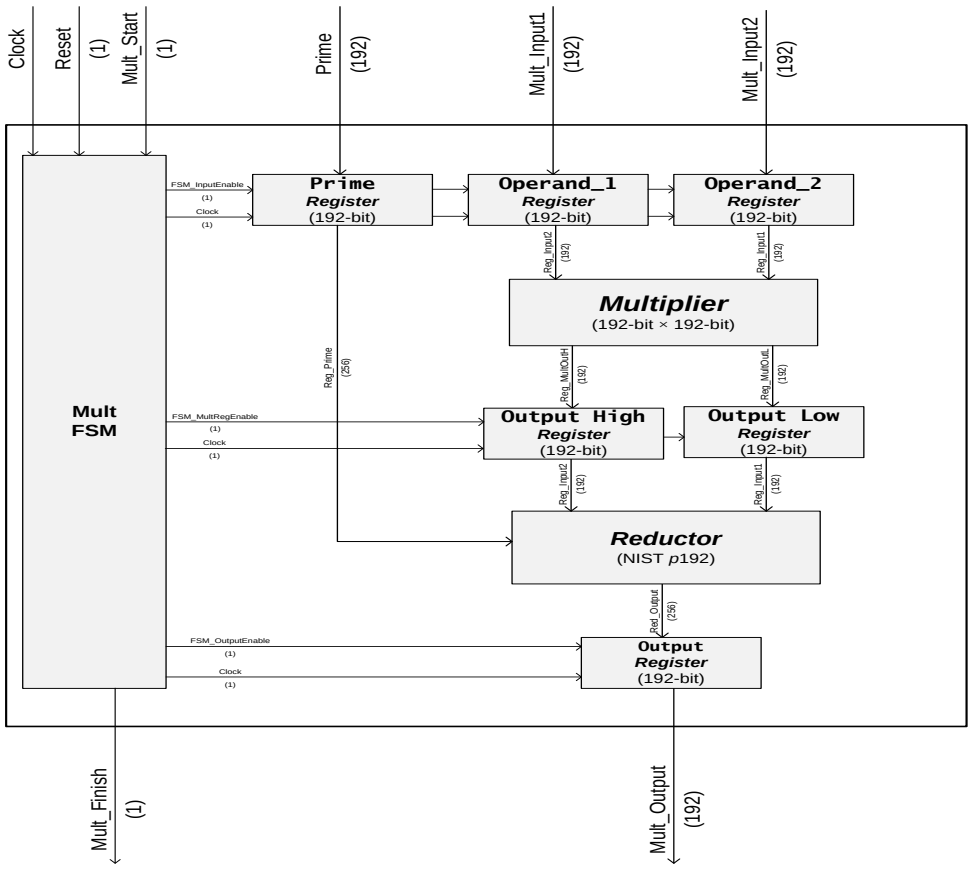

Figure 4.10: Logic Diagram of Modular Multiplication

Figure 4.10 shows the internal logic diagram of modular multiplication and reduction where we need three registers in the beginning to store the prime and the two operand of the multiplication. Then, plain[1] multiplier has been used since we just proof the concept of proposed atomic block. As result of multiplying two 192_bit inputs, we got 384_bit output. Therefore, we divide them in two registers as output low and high to perform the reduction using fast reduction modulo p_{192} recommended by NIST as shown in algorithm2.4.3. After reduction, the multiplier store the 192-bit output in register. The whole above process is controlled by the *Mult FSM* where it generating signal to communicate internally between multiplier modules and externally with *ECC FSM*.

The multiplier finite state machine consists of four states and works as shown in Figure 4.11. The multiplier will stay in state wait until it is receives $Mult_Start$ signal

[1]Plain multiplication means a star (*) operation used

from *ECC FSM*. By receiving the $Mult_Start$ signal, input registers will be loaded with input operands. In the next cycle, multiplication will be done and its result will be partitioned into low and high and stored temporary in intermediate registers as seen in figure4.10. In the following cycle, reduction will be performed and the result stores in the output register. Finally, the multiplier sends the $Mult_Finish$ signal to inform the main finite state machine *ECC FSM* that multiplication has been done.

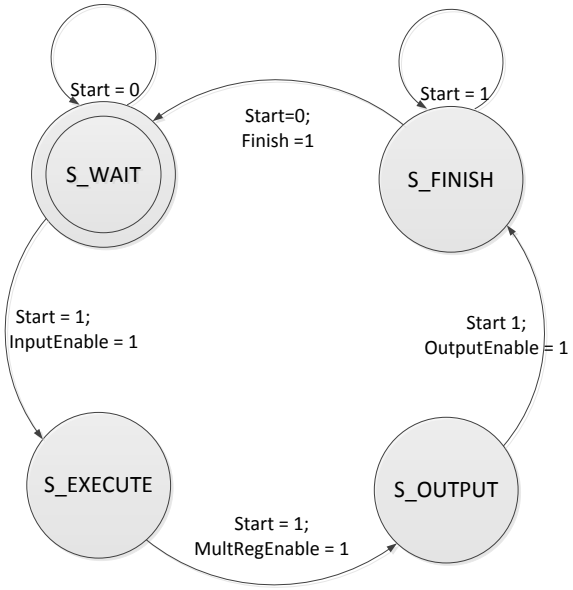

Figure 4.11: Modular Multiplier FSM

4.5 Register Allocation and Scheduling

As we see in figure 4.5, we used one multiplier and one adder to perform point operations. Therefore, we used temporary registers to store the intermediate results and retrieve them for the following operations. By investigating the point operation flowcharts, we found that at least 10 registers are required for the proposed countermeasures. Looking to the fact that some operation depends on others, we schedule field arithmetic operation of point doubling and point addition as shown in Table 4.1 and in Table 4.2 respectively. These tables are divided into to parts operation scheduling and register allocation in the register bank. The first column shows the number of iterations needed to perform point doubling and point addition. The fol-

lowing four column represent the adder and the multiplier operands' multiplexers. AS_{Type} column indicates the adder operation needed which is either zero for addition or one for subtraction. Register bank column shows how we schedule the intermediate result and retrieve them for the following operations. R_0, R_1 with R_2 have been used to store the final result before they mapped to the multiplexers in the way to be send back to the processor as we mentioned earlier.

Table 4.1: Register Allocation and Scheduling for Point Doubling

Iteration	Mux_{Mult} (Hex)		Mux_{Add} (Hex)		AS_{Type}	Register Bank									
	Op1	Op2	Op1	Op2		R0	R1	R2	R3	R4	R5	R6	R7	R8	R9
1	---	---	R0	R0	0	XP	YP	1	T1	---	---	---	---	---	---
2	R0	R0	---	---	---	T2	YP	1	T1	---	---	---	---	---	---
3	---	---	R0	R0	0	T2	YP	1	T1	T3	---	---	---	---	---
4	---	---	R1	R1	0	T2	YP	T4	T1	T3	---	---	---	---	---
5	R1	R1	---	---	---	T2	---	T4	T1	T3	T5	---	---	---	---
6	---	---	R5	R5	0	T2	T6	T4	T1	T3	T5	---	---	---	---
7	---	---	R0	R4	0	---	T6	T4	T1	T7	T5	---	---	---	---
8	R1	R1	---	---	---	---	T8	T4	T1	T7	T5	---	---	---	---
9	---	---	R1	R1	0	---	T9	T4	T1	T7	T5	---	---	---	---
10	---	---	R3	R3	0	---	T9	T4	T10	T7	T5	---	---	---	---
11	R3	R5	---	---	---	T11	T9	T4	---	T7	---	---	---	---	---
12	---	---	R3	R3	0	T11	T9	T4	T12	T7	---	---	---	---	---
13	---	---	R4	a	0	T11	T9	T4	T12	T13	---	---	---	---	---
14	R4	R4	---	---	---	T11	T9	T4	T12	T13	T14	---	---	---	---
15	---	---	R5	R3	1	T11	T9	T4	T15	T13	---	---	---	---	---
16	---	---	R0	R3	1	T11	T9	T4	T15	T13	T16	---	---	---	---
17	R4	R5	---	---	---	T11	T9	T4	T15	T17	---	---	---	---	---
18	---	---	R4	R1	1	XP-u = T11	YP-u= T9	Z = T4	XQ_new= T15	YQ_new= T18	---	---	---	---	---

Table 4.2: Register Allocation and Scheduling for Point Addition

Iteration	Mux_{Mult} (Hex)		Mux_{Add} (Hex)		AS_{Type}	Register Bank									
	Op1	Op2	Op1	Op2		R0	R1	R2	R3	R4	R5	R6	R7	R8	R9
1	---	---	R0	R3	1	XP	YP	Z	XQ	YQ	T1	---	---	---	---
2	R5	R5	---	---	---	XP	YP	Z	XQ	YQ	T1	T2	---	---	---
3	---	---	R6	R6	0	XP	YP	Z	XQ	YQ	T1	T2	T3	---	---
4	---	---	R1	R4	1	XP	YP	Z	XQ	YQ	T1	T2	T3	T4	---
5	R8	R8	---	---	---	XP	T5	Z	XQ	YQ	T1	T2	T3	T4	---
6	---	---	R6	R6	0	XP	T5	Z	XQ	YQ	T1	T2	T3	T4	T6
7	---	---	R7	R6	1	XP	T5	Z	XQ	YQ	T1	T2	T7	T4	T6
8	R0	R7	---	---	---	T8	T5	Z	XQ	YQ	T1	T2	---	T4	T6
9	---	---	R8	R0	1	T8	T9	Z	XQ	YQ	T1	T2	---	T4	T6
10	---	---	R9	R6	1	T8	T9	Z	XQ	YQ	T1	T10	---	T4	---
11	R3	R6	---	---	---	T8	T9	Z	T11	YQ	T1	---	---	T4	---
12	---	---	R0	R3	1	T8	T9	Z	T11	YQ	T1	T12	---	T4	---
13	---	---	R8	R4	0	T8	T9	Z	T11	T13	T1	T12	---	T4	---
14	R5	R2	---	---	---	T8	T9	T14	T11	T13	---	T12	---	T4	---
15	---	---	R8	R8	0	T8	T17	T14	T11	T13	T15	T12	---	T4	---
16	---	---	R1	R3	1	T8	T17	T14	T16	T13	T15	T12	---	T4	---
17	R4	R6	---	---	---	T8	T17	T14	T16	---	T15	---	---	T4	---
18	---	---	R0	R3	1	T8	T17	T14	T16	T18	T15	---	---	T4	---
19	---	---	R5	R8	1	T8	T17	T14	T16	T18	T19	---	---	---	---
20	R4	R5	---	---	---	T8	T17	T14	T16	T20	---	---	---	---	---
21	---	---	R4	R1	1	XP-u = T8	YP-u = T17	Z = T14	XQ_new = T16	YQ_new = T21	---	---	---	---	---

Chapter 5

Proposed countermeasures and Their Trade-offs

In this chapter, we propose two efficient techniques that support an increased level of security against simple power attacks and computing performance. To illustrate the proposed schemes, we discuss prior-art operations that can be subject to a power attack and their behaviour in order extract the cryptographic key. As previously discussed, the scalar multiplication is the attackers' target, where the scalar multiplication in binary method performs a point doubling operation when the scanned bit is zero. Point doubling is followed by point addition when the key bit is zero. Therefore, the attackers will be able to extract the whole key by observing the power trace and distinguishing between point doubling and point addition.

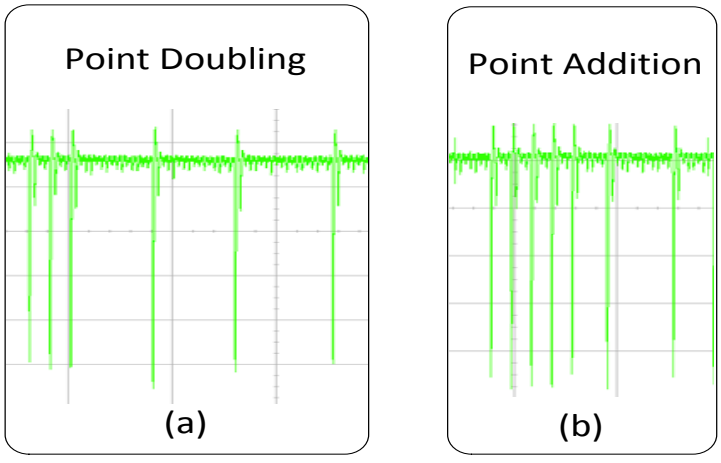

Figure 5.1: Power consumption Traces of Point Operation

Figure 5.1 shows that the power traces of point operations can be easily identified. Figure 5.1 (a) shows the trace of a point doubling operation and (b) shows the trace of a point addition; it is apparent that their shapes are very different. So, if the pattern (a) is repeated twice, that means the scanned bit of the key is "0". Also, if the pattern (a) is followed by the pattern (b), that means the scanned bit of the key is "1". By repeating this analysis to the entire power trace, the whole cryptographic can be retrieved.

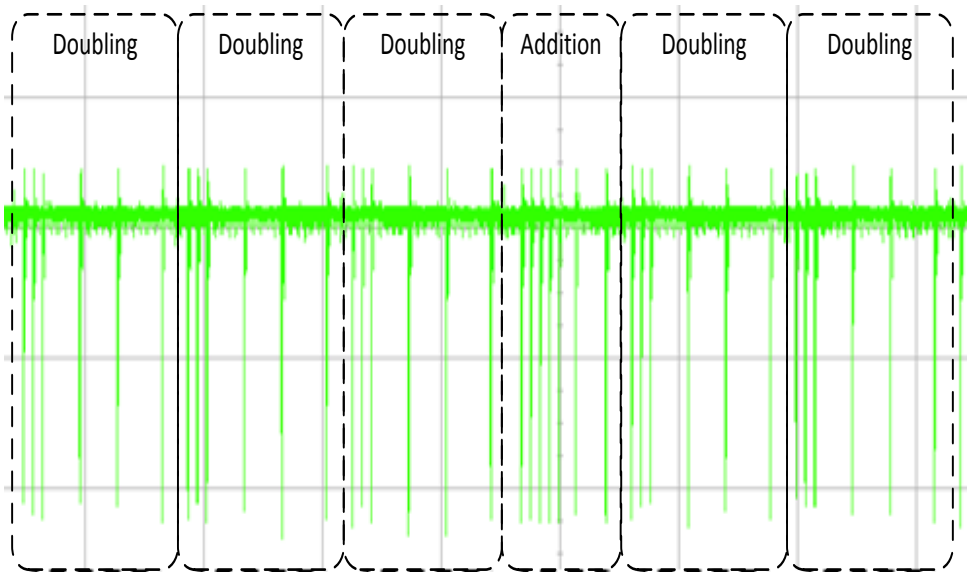

Figure 5.2: A Window of Power consumption Trace of Scalar Multiplication

As an example, Figure 5.2 shows the scalar multiplication power consumption while processing an unknown key. By analyzing the power consumption trace, the key can be easily obtained. From this figure, the first pattern corresponds to point doubling trace. This pattern is followed twice by an identical pattern. A doubling trace is followed by an addition trace. Then, two doubling traces follow the addition pattern. As result of this pattern (DDDADD), the key performed during this window is "00100". This method can be applied to extract the whole key.

In this dissertation, we propose two efficient countermeasures against simple power attacks. In the first technique, we proposed to randomize the execution order of

independent field arithmetic operation in order to generate a large number of different traces for each point operation; this technique is discussed in the following section. In the second technique, we propose an atomic block of field operation to level the power consumption of point operations; this method is presented 5.3.

5.1 Harden SPAs by Randomizing Field Arithmetic

In order to improve the robustness against SPA, we propose to shuffle independent field arithmetic operations. By changing their order of execution, a large set of point operation flowcharts will be obtained. As result, different power consumption traces to each point operation are possible at every iteration.

Figure 5.3 shows the point doubling flowchart that consists of a number of different field operations (blocks in shaded gray). In this figure, it is apparent that there is a true dependency between some of the field operations. Other operations exhibit no true dependencies. In this approach, operation with no true dependency can be shuffled. For example, operation (1) and (2) can be shuffled, while operation (1) and (4) cannot be shuffled because the output of operation (1) is used in operation (4). As result of shuffling truly independent operations, more than 60 flowcharts for point doubling with different power consumption traces can be obtained. In the implementation, we can use pseudo number generator (PNG) to decide what point operation flowchart will be active at every iteration.

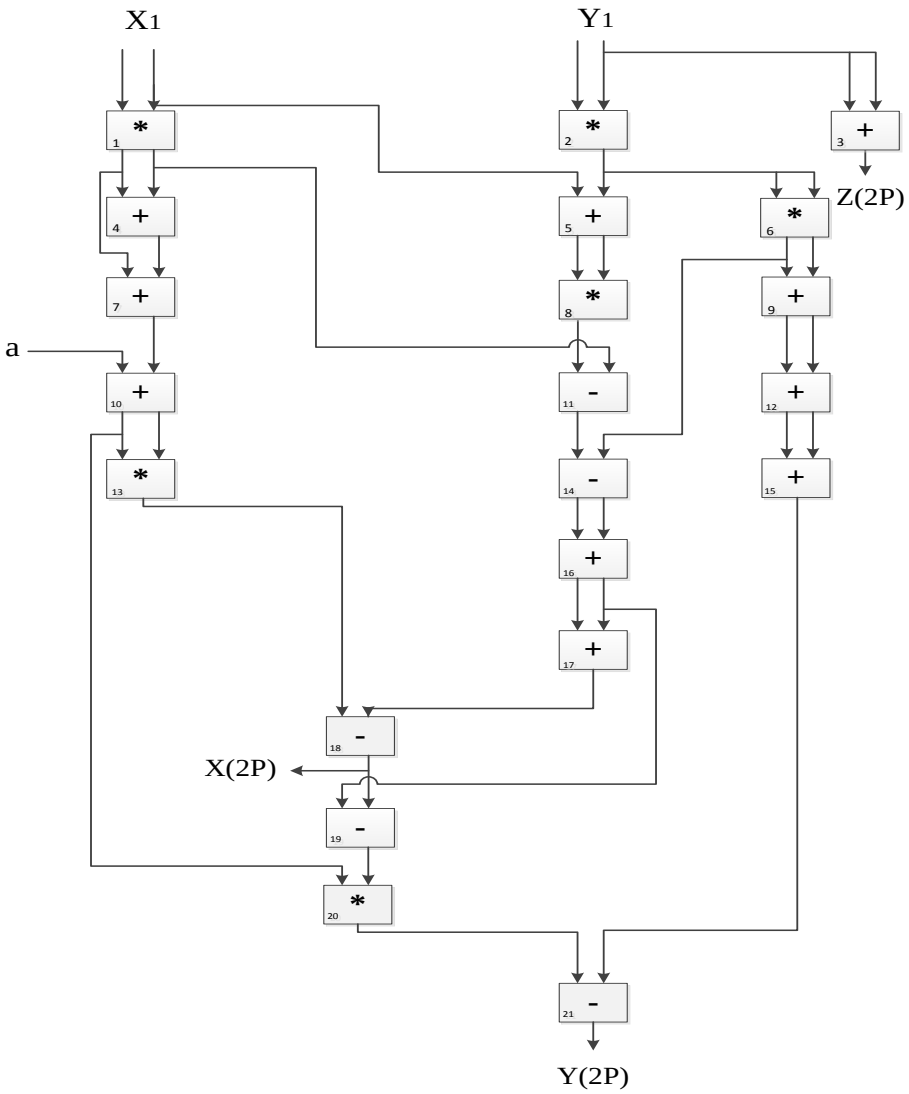

Figure 5.3: Point Doubling in Co-Z Flowchart (X_1 and Y_1 are the input arguments)

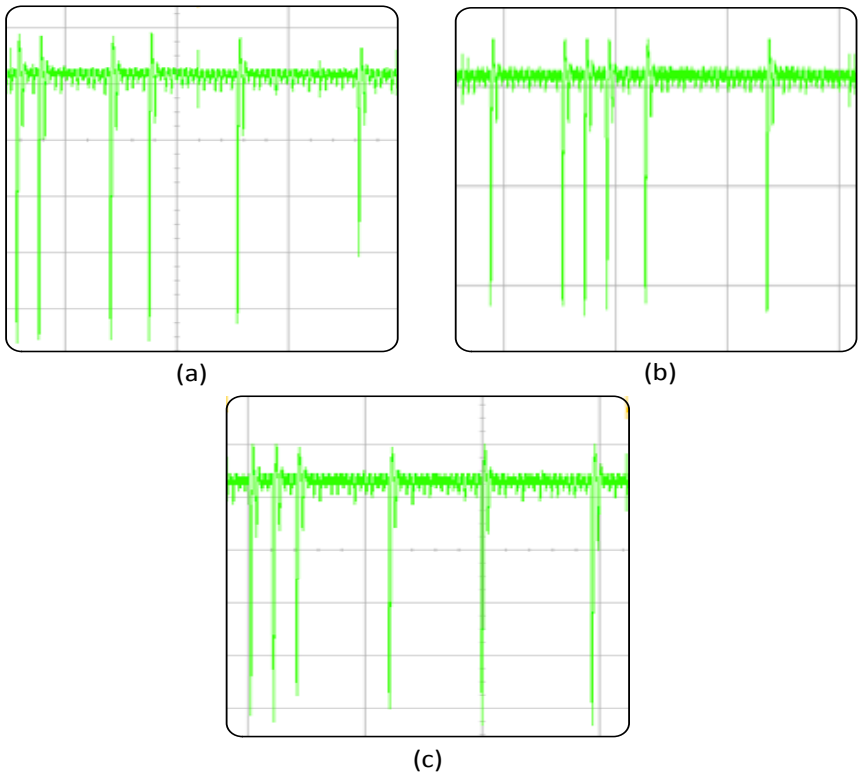

Figure 5.4: Point Doubling Power consumption traces

Figure 5.4 shows three different power consumption traces. It is apparent that they look totally different. Consequently, we increase the time needed to break the system. An attacker needs about $60^{(l)}$ attempts, where l is the length of the cryptographic key. This method comes with no extra penalty in terms of computation cost and with a minimum penalty in hardware that is needed to implement the PNG and switches.

5.2 Protected Point Operations using Atomic Block

An atomic block is a group of small number of field arithmetic operations that can be used to implement the whole algorithm. This is more efficient in terms of computation cost because it reduces the number of dummy point operations in Algorithm 3.2.1.

5.2.1 Related Work

The side channel atomicity has been introduced to elliptic curve point operations by Chevallier Mames at al. in 2004 [6]. Their basic atomic block comprizes multiplication, addition, negation and addition, *[M-A-N-A]*. The entire algorithm can be executed in a series of such atomic blocks. Two years later, Mishra used same atomic block to protect the point operations against power attacks; he enhanced the idea by pipelining the point operations [22]. In [18], the author introduced a new *[M-N-A-M-N-A-A]* atomic block. His proposed atomic block reduces the computation cost if the power consumption of squaring operation is less than the power consumption of multiplication ($S = 0.8M$).
We take the advantage of new co-Z arithmetic with update, which uses Jacobean coordinate system with the same z-coordinate. This arithmetic has been introduced in [20] and has been further explored in [25]. Considering the advantage of co-z arithmetic, we propose an atomic block that consists of two addition and one multiplication operations. The atomic is organized as [A-M-A], (addition, multiplication and addition). Our atomic block provides a number of advantages:

- It reduces the computation cost.
- It requires no extra modular multiplication.
- It requires no dummy field operation for point operations.
- It increases the level of parallelism.

5.2.2 Methodology

We investigate the point operation in co-Z with update, which is explored in [25], in order to improve our atomic block. As result, we innovate the [A-M-A] atomic block that has the advantages early discussed. For point doubling, we modified the algorithm 15 to be better protected against SPA by executing a series of the proposed atomic blocks.

Algorithm 15 Co-Z Point Doubling with Update[25]

Input: $P = (X_1 : Y_1 : 1)$

Output: $(R, P) \leftarrow DBLU(P)$ where $R \leftarrow 2P = (X_2 : Y_2 : Z_2)$ and $P \leftarrow (\lambda^2 X_1 : \lambda^3 Y_1 : \lambda)$ with $\lambda = Z_2$

$T_0 = a, T_1 = X_1, T_2 = Y_1$
$T_3 \leftarrow 2T_2$
$T_2 \leftarrow T_2^2$
$T_4 \leftarrow T_1 + T_2$
$T_4 \leftarrow T_4^2$
$T_5 \leftarrow T_1^2$
$T_4 \leftarrow T_4 - T_5$
$T_2 \leftarrow T_2^2$
$T_4 \leftarrow T_4 - T_2$
$T_1 \leftarrow 2T_4$
$T_0 \leftarrow T_0 + T_5$
$T_5 \leftarrow 2T_5$
$T_0 \leftarrow T_0 + T_5$
$T_4 \leftarrow T_0^2$
$T_5 \leftarrow 2T_1$
$T_4 \leftarrow T_4 - T_5$
$T_2 \leftarrow 8T_2$
$T_5 \leftarrow T_1 - T_4$
$T_5 \leftarrow T_5 * T_0$
$T_5 \leftarrow T_5 - T_2$
return $R = (T_4 : T_5 : T_3)$, $P = (T_1 : T_2 : T_3)$

Algorithm 15 shows point doubling in co-z arithmetic and illustrates that updating the value of point P is performed without extra operations [20]. To make this algorithm protected against SPA, we introduce a new atomic block to it, as it is shown in Algorithm 16.

Algorithm 16 Proposed Protected Co-Z Point Doubling with Update

Input: $P = (X_1 : Y_1 : 1)$

Output: $(R, P) \leftarrow DBLU(P)$ where $R \leftarrow 2P = (X_2 : Y_2 : Z_2)$ and $P \leftarrow (\lambda^2 X_1 : \lambda^3 Y_1 : \lambda)$ with $\lambda = Z_2$

$R_0 = X_1, R_1 = Y_1, R_2 = Z = 1$
$R_3 \leftarrow 2R_0$
$R_0 \leftarrow R_0^2$
$R_4 \leftarrow 2R_0$
$R_2 \leftarrow 2R_1$
$R_5 \leftarrow R_1^2$
$R_1 \leftarrow 2R_5$
$R_4 \leftarrow R_0 + R_4$
$R_1 \leftarrow R_1^2$
$R_1 \leftarrow 2R_1$
$R_3 \leftarrow 2R_3$
$R_0 \leftarrow R_3 * R_5$
$R_3 \leftarrow 2R_3$
$R_4 \leftarrow R_4 + a$
$R_5 \leftarrow R_4^2$
$R_3 \leftarrow R_5 - R_3$
$R_5 \leftarrow R_0 - R_3$
$R_4 \leftarrow R_4 * R_5$
$R_4 \leftarrow R_4 - R_1$
return $R = (R_3 : R_4 : R_2)$, $P = (R_0 : R_1 : R_2)$

In term of hardware cost, we consider that squaring is equal to multiplication ($S = M$) because both are performed by a single component in hardware. Algorithm 16 shows the proposed protected co-z point doubling using atomic block; it is based on three field arithmetic operations addition/subtraction, multiplication/squaring, and addition/subtraction, [A-M-A]. This atomic block has a specific power consumption trace. However, this pattern will be repeated to perform either point doubling or point addition. Therefore, the attacker will not be able to extract the key bit from the power trace.

Figure 5.5 shows the flowchart of the protected point doubling operation. This flowchart uses the proposed atomic block, and requires 6 sequential atomic block to

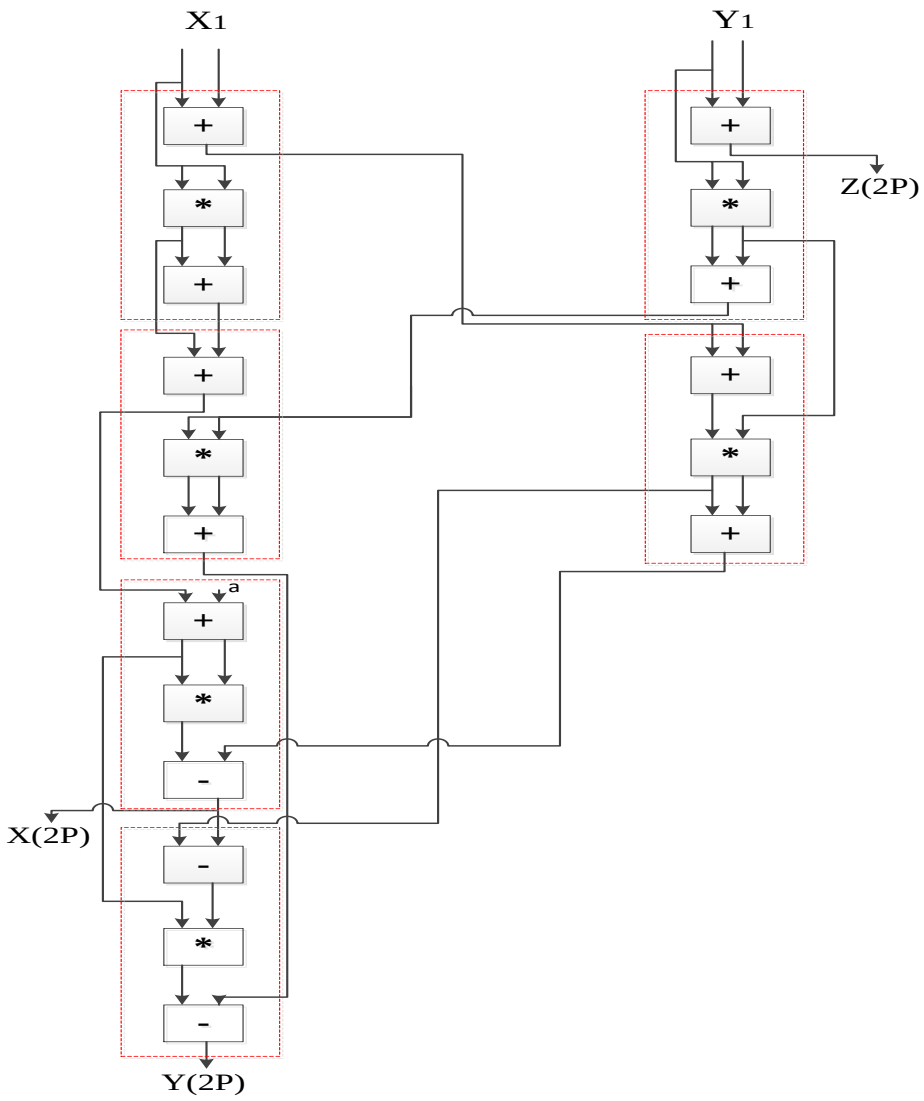

Figure 5.5: Point Doubling using atomic block

perform point doubling. A second main advantage of this flowchart is the intrinsic protection against fault attacks since it executes no dummy operations.

Likewise, we modified co-z point addition with update shown in [25] to be secured against SPA. The following algorithm 17 shows same atomic block used for point addition; the possibility of distinguishing between the point operations is now eliminated .

Algorithm 17 Proposed Protected Co-Z Point Addition with Update

Input: $P = (X_1 : Y_1 : Z)$, $Q = (X_2 : Y_2 : Z)$

Output: $(R, P) \leftarrow DBLU(P)$ where $R \leftarrow 2P = (X_2 : Y_2 : Z_2)$ and $P \leftarrow (\lambda^2 X_1 : \lambda^3 Y_1 : \lambda)$ with $\lambda = Z_2$

$R_0 = X_1, R_1 = Y_1, R_2 = Z, R_3 = X_2, R_4 = Y_2$
$R_5 \leftarrow R_0 - R_3$
$R_6 \leftarrow R_5^2$
$R_7 \leftarrow 2R_6$
$R_8 \leftarrow R_1 - R_4$
$R_1 \leftarrow R_8^2$
$R_9 \leftarrow 2R_6$
$R_7 \leftarrow R_7 - R_6$
$R_0 \leftarrow R_0 * R_7$
$R_1 \leftarrow R_8 - R_0$
$R_6 \leftarrow R_9 - R_6$
$R_3 \leftarrow R_3 * R_6$
$R_6 \leftarrow R_0 - R_3$
$R_4 \leftarrow R_8 + R_4$
$R_2 \leftarrow R_5 * R_2$
$R_5 \leftarrow 2R_8$
$R_3 \leftarrow R_1 - R_3$
$R_1 \leftarrow R_4 * R_6$
$R_4 \leftarrow R_0 - R_3$
$R_5 \leftarrow R_5 - R_8$
$R_4 \leftarrow R_4 * R_5$
$R_4 \leftarrow R_4 - R_1$
return $R = (R_3 : R_4 : R_2)$, $P = (R_0 : R_1 : R_2)$

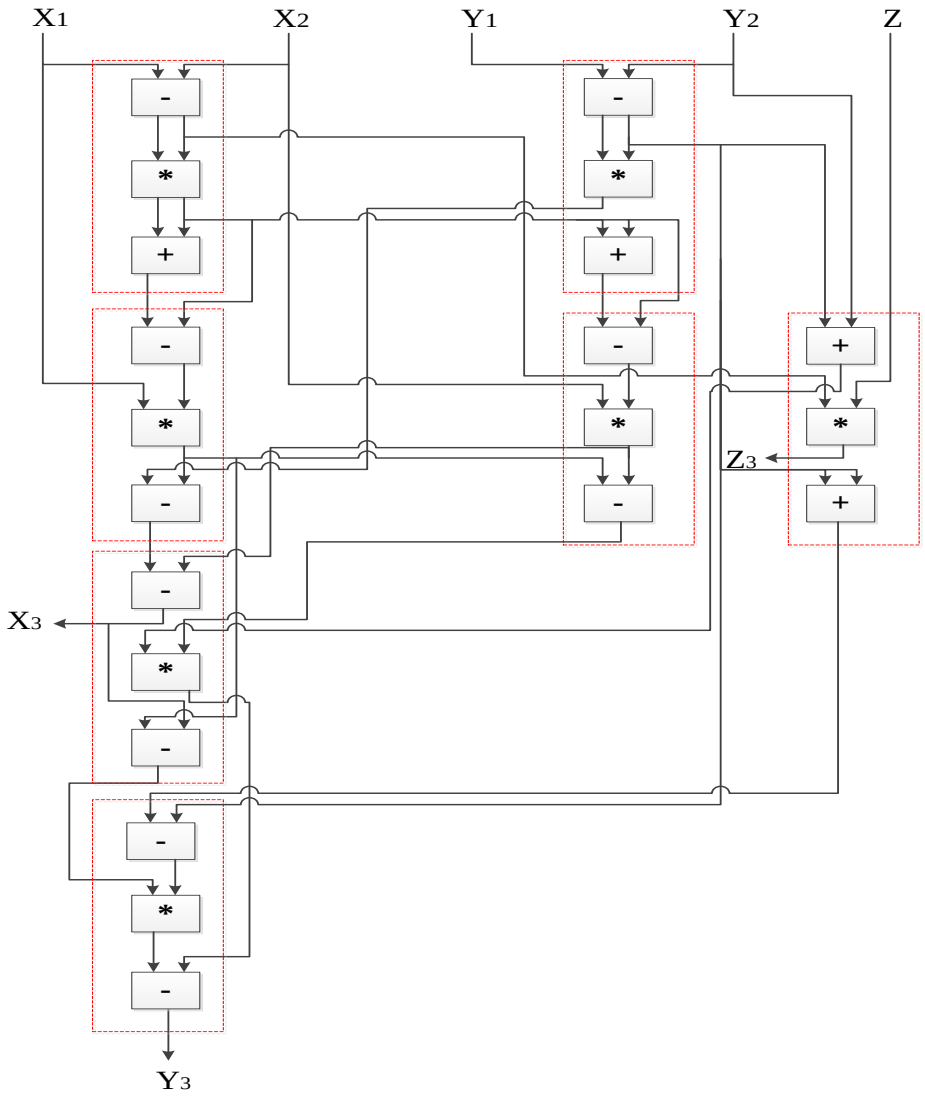

Figure 5.6: Point Addition using atomic block

Similarly, figure 5.6 shows that the proposed atomic block performs the protected point addition with 7 atomic blocks; protection against SPA and fault attacks is intrinsically provided.

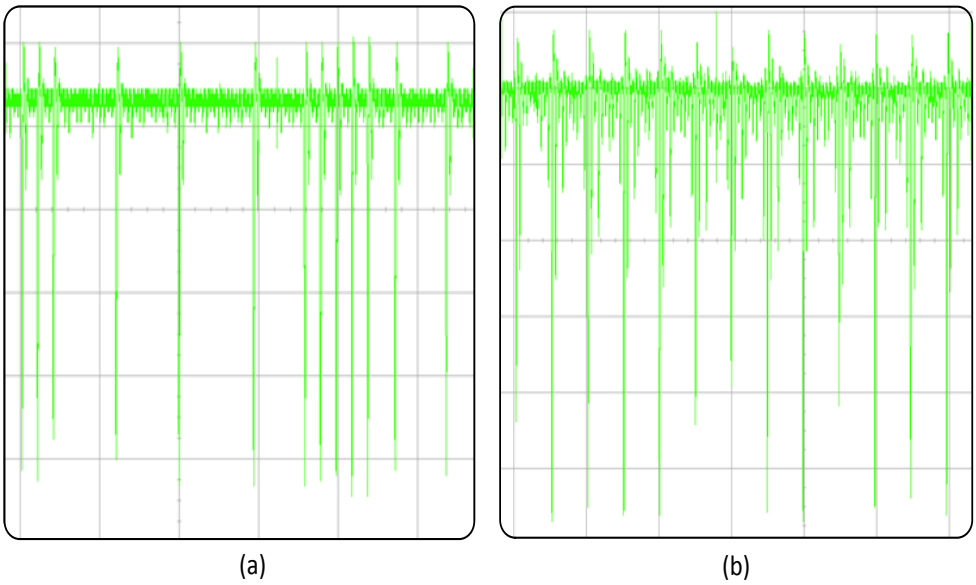

Figure 5.7: Power Consumption Trace of Protected and Unprotected Point Operation

Figure 5.7 shows protected and unprotected scalar multiplication, where point doubling followed by point addition is performed in both cases. In this figure, (a) shows the power consumption trace of unprotected scalar multiplication; it is apparent that the two patterns are different. This means the processed key bit is "1". In comparison, (b) shows the power consumption trace of protected scalar multiplication. Here, it is much more difficult to distinguish between point doubling and point addition. Figure 5.7 (b) shows a periodic pattern that occurs during the process of the scalar multiplication. As result, the attackers will not be able to predict the scanned bit. As a result, by using the proposed atomic block, this cryptosystem exhibit a better protection against side channel attacks(SPAs). In addition, it is more efficient in terms of computation cost since it use extra field operations rather than extra point operations.

5.2.3 Performance Comparison

First, let start with the atomic block [M-A-N-A] which is introduced by Chevallier-Mames at al. in [6]. Performing point doubling needs 10 atomic blocks, whereas point addition costs 16 atomic blocks. Therefore, the cost of point operations are $16M+32A$ for a point addition, and $10M+22A$ for a point doubling. Point addition was enhanced by Mishra in [22], where he reduced the cost to $11M + 22A$. Then, Patrick [18] proposed a new atomic block [M-N-A-M-N-A-A] that outperforms the original atomic block. This atomic block reduced the cost of point doubling to $8M + 12A$. However, the point addition costs $12M + 18A$. Our proposed atomic block [A-M-A], which takes the advantages of co-z representation, outperforms the previous work as it is illustrated in Table 5.1.

Table 5.1: Performance Comparison of Atomic Blocks

Works	Point Operation	
	Point Doubling	Point Addition
Proposed	$6M + 12A$	$7M + 14A$
Patrick [18]	$8M + 12A$	$12M + 18A$
Mishra [22]	$10M + 22A$	$11M + 22A$
Chevallier-Mames at al[6]	$10M + 22A$	$16M + 32A$

Table 5.1 summarizes the computation cost of varies atomic blocks of point doubling and addition operation. These atomic blocks are based on different set of field operation. First, [M-A-N-A] is the atomic block introduced by Chevallier-Mames at al. in [6] and used by Mishra in [22]. In addition, [M-N-A-M-N-A-A] has been presented by Patrick Longa in [18]. Finally, [A-M-A] is the atomic block proposed in this work. In this dissertation, improvement in term of computation cost is achieved; it ranges from 25% to 45% for point doubling and from 40% to 56% for point addition in comparison to [6], [22], and [18].

Furthermore, [25] shows that the cost of point operations using the Joy's *Always Double-and-Add* algorithm is $9M + 7S$. This algorithm might be subject to fault attacks because it executes dummy point operations. To our knowledge, it is not efficient in term of the computation cost, too. To illustrate that, Lets call equation 2.4 to show the cost of point operation if the key is 192-bit long. Further, we assume

$M = S$:

$$\begin{aligned} Cost(kP) &= (192-1)(CostPD + CostPA); \\ Cost(kP) &= (192-1)(16M) = 3056M \end{aligned} \tag{5.1}$$

In Comparison with 5.1, our proposed atomic block uses the classic binary algorithm to calculate the scalar multiplication using co-z with update. This algorithms is more efficient in terms of computation cost because it performs each point operation only when needed. Moreover, point operations do not require extra heavy field operation. To illustrate that, lets analyze equation 2.3, where the cost of each operation using the proposed atomic block is $6M$ for point doubling and $7M$ for point addition:

$$\begin{aligned} Cost(kP) &= (l-1)CostPD + (l/2)CostPA; \\ Cost(kP) &= (191)(6M) + (96)(7M) = 1818M \end{aligned} \tag{5.2}$$

From 5.1 and 5.2, we see that using the proposed protected point operation has reduced the computation cost to about 41%. In addition, it also safe to error attacks since neither dummy field operations nor point operations are performed.

5.3 Protected Parallel Point Operation

In the previous section, we have explored old and proposed atomic blocks that perform point doubling and point addition. Then, we have presented the differences in term of performance where the proposed atomic block is more efficient.In this section, we show different parallel architecture mechanisms and how the proposed protected point operation, that mentioned in the previous section, supports parallelism.

5.3.1 Related Work

Security and performance play major factors in the implementation of cryptosystems. Thus, with the effort of securing the implementation in last section, enhancing the level of parallelism is required. We found several secure parallel schemes introduced to the elliptic curve at different mathematical layers, shown in figure 1.1.
In [13], the authors shows that point doubling and point addition run in parallel at every iteration. On the other hand, [10] parallelizes point operation at field arithmetic

operations, where a number of field operation run in parallel. However, both [13] and [10] use this parallel schemes on special curve over prime field using a Montgomery ladder, which is naturally protected against simple side channel attacks because it performs point doubling and addition at every iteration.

In [22], the authors used the atomic block proposed in [6] to protect their pipelined point operation, where up to two atomic block runs in parallel. [1] parallelizes the point operation at the field arithmetic level using atomic block for better protection against simple power analysis.

5.3.2 Methodology

An elliptic curve cryptosystem can be mapped to a parallel architecture at the different mathematical levels shown in figure 1.1. At point multiplication level, point doubling and point addition can be run in parallel; however, this is not efficient in term of computation cost. Therefore, we focus our work to parallelize co-z point operations at field arithmetic level. In Figure 5.5 show that six atomic blocks are executed to implement point doubling. Each atomic block has three field arithmetic operations; addition, multiplication and addition. In this figure, it is apparent that each horizontal pair of atomic blocks on point doubling can be run in parallel. Atomic blocks (1) and (2) run in parallel.

Likewise, point addition can be mapped onto a parallel architecture at atomic level. However, it requires an extra atomic block with a number of dummy operations.

Figure 5.8 shows that each horizontal pair of atomic blocks runs in parallel. As you can see from this figure, there are number of dark operations with no operands. These operation are dummy in order to support the atomic block structure. Point doubling needs more investigation to increase its efficiency.

5.3.3 Performance Comparison

In [10], the author presented a parallel scheme using a Montgomery ladder which is naturally protected against simple power analysis. The cost of each parallel execution of point operations is ten field multiplication. Montgomery ladder method needs (l-1) iteration to perform l-bit scalar multiplication. If $l = 192$ bit, the total cost of this

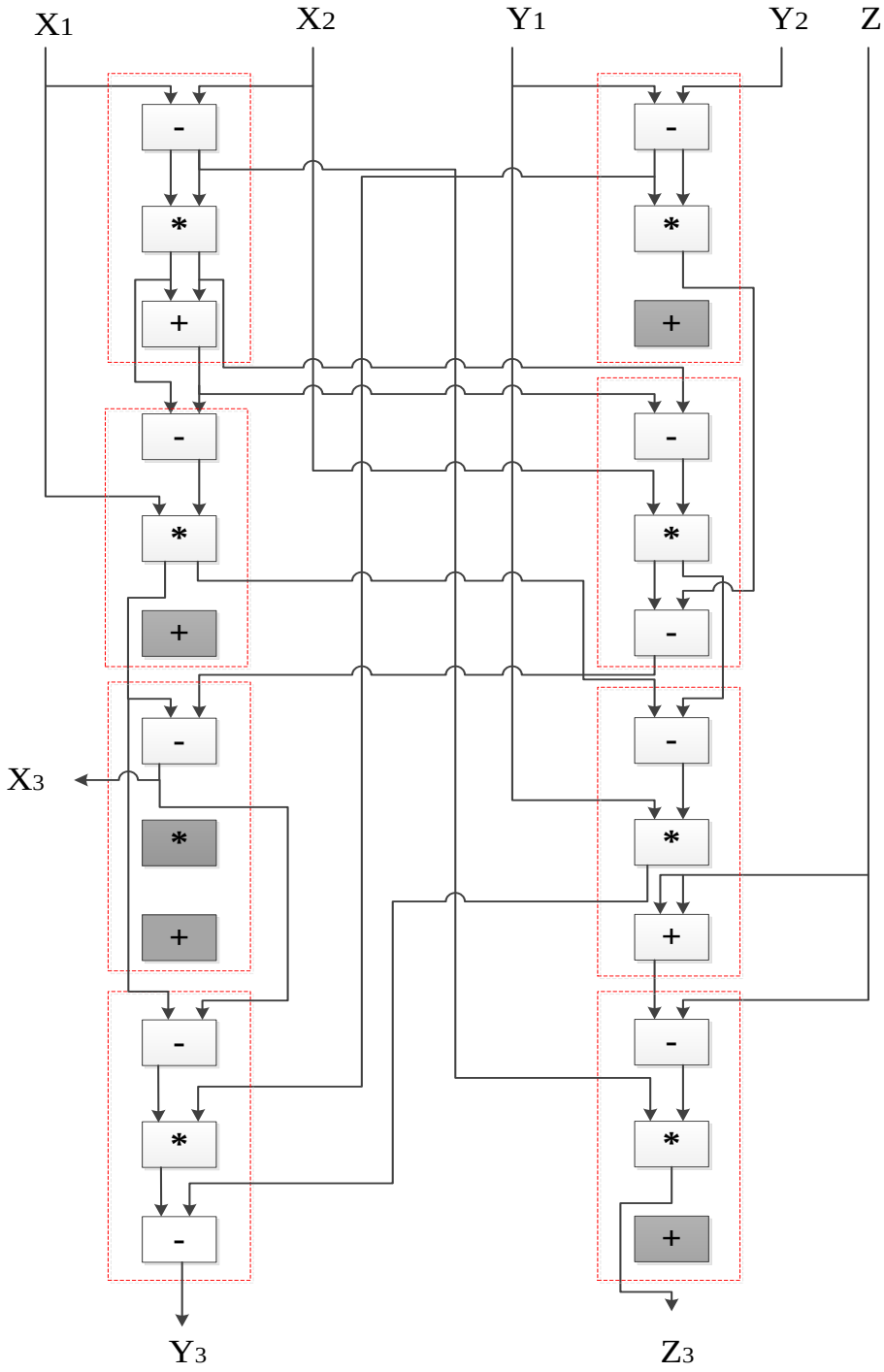

Figure 5.8: Parallel Protected Point Addition

approach is $1910M$. In [13], the cost of both point doubling and addition formulae are $9M$ and $10M$, respectively. In this article, the author fixes the cost at every iteration to $10MS$, which is the cost of point doubling. A single extra point doubling is needed to complete the computation of the scalar multiplication using the Montgomery ladder. Thus, the cost of 192-bit scalar is $(9M) + (l-1)(10M) = 1919$. The pipelined scheme presented by Mishra in [22] reduces the cost to $1726M$ for binary algorithm. It is reduced the cost to $1534M$ by using NAF algorithm. In [18], the author proposed a parallel scheme using his proposed atomic block to be protected against SPA. Point doubling requires two parallel atomic blocks while four atomic blocks are needed for point addition. Since the author used NAF algorithm, his scheme costs $2(l-1)$ atomic blocks for doubling and $4(l/3)$ atomic blocks for addition. If we consider the scalar is 192-bit, the total cost is $1276M$.

In this work, two atomic blocks run in parallel. Four atomic blocks are needed to perform each point operations. Thus, the total cost of 192-bit binary scalar multiplication is $4(n-1)M + 4(n/2) = 1148M$. If NAF algorithm is used, the cost is reduced to $1020M$.

Table 5.2: Performance Comparison of Parallel Protected Point Operations for $l = 192$

Work	Algorithm	Cost
Proposed	NAF	1020 M
Patrick [18]	NAF	1276 M
Mishra[22]	NAF	1534 M
Fischer [10]	Montgomery ladder	1910 M
Izu and Takagi[13]	Montgomery ladder	1919 M

Table 5.2 summarizes most parallel protected point operations with their cost. It is apparent that our proposed parallel scheme exhibits speed in comparison to prior-art. By this scheme, reduction in term of computation time ranging from 20% to 47% is achieved. To conclude, we enhance the level of parallelism on co-z point operation and reduce the computation time by using the proposed atomic block.

Chapter 6

Conclusions

In this dissertation, we have proposed three efficient techniques to design point operations in co-Z representation that supports an increased level of security against simple power attacks, and provide a higher computation performance. In the first technique, we have proposed to randomize independent the field arithmetic operations within each point operation. The main idea is to generate a large number of different flowcharts for each point operation, namely 60. At every iteration, this polymorphic implementation can use one out of 60 different flowcharts. These flowcharts are randomly chosen using a pseudo number generator (PNG). For 192-bit key (scalar), the number of attempts to detect the pattern of each point operation is increased by 60^{192}; this require a very large time for a full search of the space making this approach infeasible.

In the second technique, we have proposed an atomic block that consists of addition, multiplication, and addition [A-M-A]. This atomic block has several advantages:

- It reduces the computation cost.
- It requires no extra modular multiplication involves.
- It requires no dummy field operation for point operations.
- It increases the level of parallelism.

By using this technique, we have protected the implementation against SPA. We gain an improvement in terms of computation time with respect to other existing works, ranging from 25% to 45% for point doubling and from 40% to 56% for point addition.

Finally, we have proposed an implementation that enhances the level of parallelism of protected point operations in the atomic block approach. Our main idea is to parallelize the atomic operations within a point operation. Point doubling operation is already an adopted parallel architecture; however, point addition needs an extra atomic block with more dummy operations. By this scheme, a reduction in term of computation time ranging from 20% to 47% is achieved.

To proof the concept, we have implemented the scalar multiplication on the largest chip in SPARTAN 6 family, which is "***XC6SLX150T***". We have downloaded our implementation on FPGA in order to capture the power consumption trace while processing the scalar multiplication. It was apparent that the prior art scalar multiplication was unprotected against side channel attacks. Then, we have applied the proposed techniques and showed how they are effective in term of system security and performance. More information about used experimental set up can be found in the appendix.

To sum up, we have provided the customers with three schemes to counter the simple power analysis. The first scheme provides an improved security at no hardware cost; the second technique provides protection against simple power analysis with extra modular addition operations; the third scheme provides both security and speed by adopting the second techniques and enhancing the level of parallelism.

6.1 Future Work

In this dissertation, our implementation has several limitations. Therefore, further technical investigation can be derived from this work at different levels as follows:

At Scalar Multiplication level: To prove the concept, we used a classic binary method to perform the scalar multiplication. It is equally important to support *NAF* and *w-NAF* algorithms in order to enhance the performance with the enhanced system security. Due to the limited capacity of a high FPGA device, we implement the scalar multiplication over prime field using one of the NIST recommended prime which is p-192. It would be interesting to investigate all recommended primes.

At Point Operation Level: We proposed protected parallel point operations. Point doubling is adopted parallel architecture with no extra atomic block. In order to have a point addition that supports parallel execution, an atomic block with

extra dummy operations is needed. Therefore, more investigation can be done to optimize the point addition.

In addition, we used a pseudo number generator to randomly choose the flowchart of each operation at every iteration. To further increase the security, other random number generators could be used. For example, it would be interesting to investigate a random number generator based on thermal noise.

At Field Arithmetic Level: We use a series of atomic blocks of field operations to perform point operation. By observing the power consumption trace, we noticed that the modular multiplication consumed a large power in comparison to modular addition. Moreover, this approach uses a large area in the FPGA. It is a point of interest to investigate how to improve the modular multiplication.

At Circuit Level: It would be interesting to investigate the behaviour of the scalar multiplication at the circuit level in order to level the power consumption. For example, we can increase the level of parallelism at architecture level and use a very low power logic family.

Bibliography

[1] Turki F. Al-Somani. Overlapped parallel computations of scalar multiplication with resistance against side channel attacks. *International Journal of Information and Computer Security (IJICS)*, 2(3):261 –278, 2008.

[2] Hamad Alrimeih. Fast and flexible hardware support for elliptic curve cryptography over multiple standard prime finite fields. 2012, 2012.

[3] Elaine Barker, William Barker, William Burr, William Polk, and Miles Smid. Recommendation for key management - part 1: General (revision 3). In *NIST Special Publication*, July 2012.

[4] Paul Barrett. Implementing the rivest shamir and adleman public key encryption algorithm on a standard digital signal processor. In *Proceedings on Advances in cryptology—CRYPTO '86*, pages 311–323, London, UK, UK, 1987. Springer-Verlag.

[5] I. Blake, G. Seroussi, N. Smart, and J. W. S. Cassels. *Advances in Elliptic Curve Cryptography (London Mathematical Society Lecture Note Series)*. Cambridge University Press, New York, NY, USA, 2005.

[6] B. Chevallier-Mames, M. Ciet, and M. Joye. Low-cost solutions for preventing simple side-channel analysis: side-channel atomicity. *Computers, IEEE Transactions on*, 53(6):760 – 768, june 2004.

[7] Henri Cohen and Gerhard Frey, editors. *Handbook of elliptic and hyperelliptic curve cryptography*. CRC Press, 2005.

[8] Jean-Sébastien Coron. Resistance against differential power analysis for elliptic curve cryptosystems. pages 292–302, 1999.

[9] George I. Davida and Yair Frankel, editors. *Information Security, 4th International Conference, ISC 2001, Malaga, Spain, October 1-3, 2001, Proceedings*, volume 2200 of *Lecture Notes in Computer Science*. Springer, 2001.

[10] Wieland Fischer, Christophe Giraud, Erik Woodward Knudsen, and Jean-Pierre Seifert. Parallel scalar multiplication on general elliptic curves over fp hedged against non-differential side-channel attacks. *IACR Cryptology ePrint Archive*, 2002:7, 2002.

[11] Darrel R. Hankerson, Alfred J. Menezes, and Scott A. Vanstone. *Guide to elliptic curve cryptography*. Springer professional computing. Springer, New York, 2004.

[12] Tetsuya Izu and Tsuyoshi Takagi. Fast elliptic curve multiplications with simd operations. In *ICICS*, pages 217–230, 2002.

[13] Tetsuya Izu and Tsuyoshi Takagi. A fast parallel elliptic curve multiplication resistant against side channel attacks. In *Public Key Cryptography*, pages 280–296, 2002.

[14] M. Joye. Elliptic curves and side-channel analysis. *ST Journal of System Research*, 4(1):283–306, January 2003.

[15] Neal Koblitz. Elliptic curve cryptosystem. *Mathematics of Computation*, 48(177):203–209, 1987.

[16] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. pages 388–397. Springer-Verlag, 1999.

[17] Patrick Longa. Accelerating the scalar multiplication on elliptic curve cryptosystems over prime fields. *IACR Cryptology ePrint Archive*, 2008:100, 2008.

[18] Patrick Longa and Ali Miri. Fast and flexible elliptic curve point arithmetic over prime fields. *Computers, IEEE Transactions on*, 57(3):289 –302, march 2008.

[19] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards (Advances in Information Security)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[20] Nicolas Meloni. New point addition formulae for ecc applications. In *Proceedings of the 1st international workshop on Arithmetic of Finite Fields*, WAIFI '07, pages 189–201, Berlin, Heidelberg, 2007. Springer-Verlag.

[21] Victor S. Miller. Use of elliptic curves in cryptography. In *CRYPTO*, pages 417–426, 1985.

[22] P.M. Mishra. Pipelined computation of scalar multiplication in elliptic curve cryptosystems (extended version). *Computers, IEEE Transactions on*, 55(8):1000 –1010, Aug. 2006.

[23] Peter L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, 1985.

[24] National Institute of Standards and Technology. *FIPS PUB 186-2: Digital Signature Standard (DSS)*. January 2000.

[25] Atsuko Miyaji Matthieu Rivain Raveen R. Goundar, Marc Joye and Alexandre Venelli. Scalar multiplication on weierstrab elliptic curves from co-z arithmetic. *Cryptographic Engineering*, 1(2):161–176, 2011.

[26] R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126, 1978.

[27] J.E. Robertson. A new class of digital division methods. *Electronic Computers, IRE Transactions on*, EC-7(3):218–222, 1958.

[28] Francisco Rodríguez-Henríquez, N. A. Saqib, A. Díaz-Pèrez, and Cetin Kaya Koc. *Cryptographic Algorithms on Reconfigurable Hardware (Signals and Communication Technology)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[29] Babak Zakeri. On studying whitenoise stream-cipher against power analysis attacks. 2012, 2012.

Appendix A

Experimental Set-up

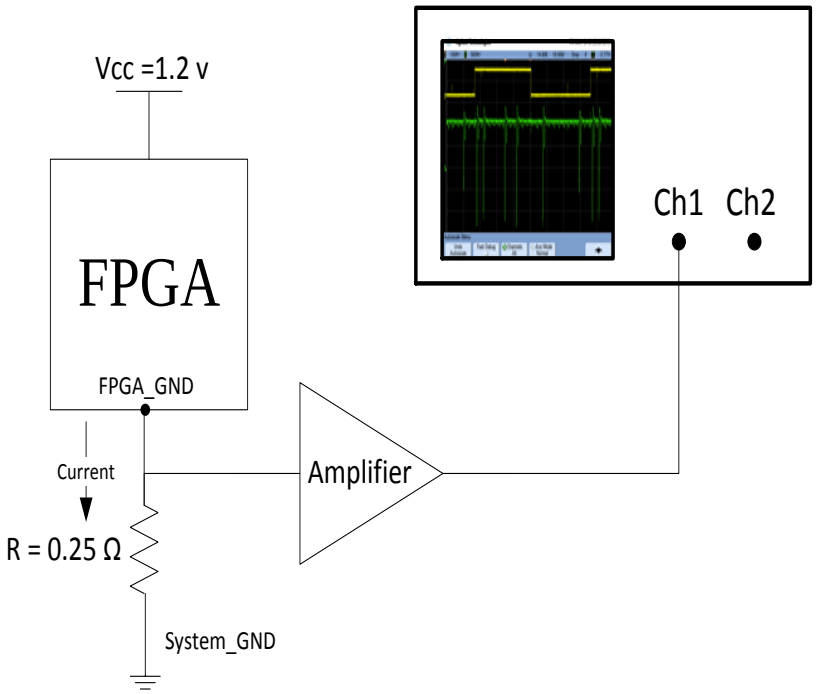

Figure A.1: Experimental Set up

Figure A.1 presents the experimental set up that we have used to capture the power consumption trace of our implementation. As it is apparent, the set up consists of an FPGA, an oscilloscope, an amplifier, and a tiny resistor. A tiny resistor is

connected between the ground of FPGA and the ground of the system. A small amount of current passes through the resistor. The power on this resistor is the power consumption of our implementation. Since the power consumption is so small, the amplifier is connected to amplify the signal to be visible. The output of the amplifier is connected to the oscilloscope to capture the power consumption trace.

In this work, we have used this board that designed by Babak Zakeri in [29]. We have downloaded our implementation through JTAG connector into the FPGA. Then, we have investigated the power consumption signal of different operation of our implementation.