10-5-2017

FPGA IMPLEMENTATION FOR ELLIPTIC CURVE CRYPTOGRAPHY OVER BINARY EXTENSION FIELD

Che Chen
University of Windsor

FPGA IMPLEMENTATION FOR ELLIPTIC CURVE CRYPTOGRAPHY OVER BINARY EXTENSION FIELD

by

Che Chen

A Thesis
Submitted to the Faculty of Graduate Studies
through the Department of Electrical and Computer Engineering
in Partial Fulfillment of the Requirements for
the Degree of Master of Applied Science
at the University of Windsor

Windsor, Ontario, Canada

2017

FPGA IMPLEMENTATION FOR ELLIPTIC CURVE CRYPTOGRAPHY OVER BINARY EXTENSION FIELD

by

Che Chen

APPROVED BY:

_____

D. Wu
School of Computer Science

_____

R. Rashidzadeh
Department of Electrical and Computer Engineering

_____

H. Wu, Advisor
Department of Electrical and Computer Engineering

7 August, 2017

AUTHOR'S DECLARATION OF ORIGINALITY

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

ABSTRACT

Elliptic curve cryptography plays a crucial role in network and communication security. However, implementation of elliptic curve cryptography, especially the implementation of scalar multiplication on an elliptic curve, faces multiple challenges. One of the main challenges is side channel attacks (SCAs). SCAs pose a real threat to the conventional implementations of scalar multiplication such as binary methods (also called doubling-and-add methods). Several scalar multiplication algorithms with countermeasures against side channel attacks have been proposed. Among them, Montgomery Powering Ladder (MPL) has been shown an effective countermeasure against simple power analysis. However, MPL is still vulnerable to certain more sophisticated side channel attacks. A recently proposed modified MPL utilizes a combination of sequence masking (SM), exponent splitting (ES) and point randomization (PR). And it has shown to be one of the best countermeasure algorithms that are immune to many sophisticated side channel attacks [11]. In this thesis, an efficient hardware architecture for this algorithm is proposed and its FPGA implementation is also presented. To our best knowledge, this is the first time that this modified MPL with SM, ES, and PR has been implemented in hardware.

DEDICATION

To my loving parents:

Father: Zhanming Chen
Mother: Chun Xu

ACKNOWLEDGEMENTS

I would like to thank my supervisor Dr. Huapeng Wu for his patient instruction on my thesis. Dr. Wu is the most knowledgably and kind professor I have ever met, and offered many invaluable opinions and advices during my study and work. I also thank Dr. Dan Wu for his feedbacks on the thesis, and Dr. Rashid Rashidzadeh for his detailed comment to improve this thesis.

Additionally, I would like to appreciate my loving parents. Without their support and encouragement, it is impossible for me to achieve such accomplishment.

I would also grateful to all my colleagues and friends for their support and time. Finally, I wish to show my gratitude to everyone at the Faculty of ECE for the help I received.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF ACRONYMS

DPA	Differential Power Analysis
ECC	Elliptic Curve Cryptosystem
ECDHKE	Elliptic Curve Diffie–Hellman Key Exchange
ECDLP	Elliptic Curve Discrete Logarithm Problem
ECDSA	Elliptic Curve Digital Signature Algorithm
ECIES	Elliptic Curve Integrated Encryption Scheme
EM	Electro Magnetic
FPGA	Field-programmable Gate Array
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuits
LUT	Look Up Table
MPL	Montgomery Powering Ladder
MVN	Multivariate Normal Distribution
RSA	Rivest, Shamir, Adleman
SCA	Side Channel Attack
SPA	Simple Power Analysis

CHAPTER I

INTRODUCTION

1.1 Motivation

The Internet is increasingly important to the people all over the world who use it for personal and business purposes. While the internet brings much convenience to people, there still exist security risks and vulnerabilities in using the internet. For example, various cyber-attacks, including side channel attacks, pose a great danger for the Internet users. Network security, which provides physical and software countermeasures to protect the network from unauthorized access and attacks, becomes a very active research area and industry. Cryptography plays a critical role in providing essential and unique network security services to the internet.

There are two main families of cryptography from the point of view of key generation, symmetric-key cryptography and asymmetric-key cryptography. In symmetric-key cryptography system, there is only one key used both for encryption and decryption. This system requires that both parties involved in the communication share one secret key, which has to be pre-arranged in advance in a procedure called key establishment. This is regarded as a main drawback of symmetric-key cryptography system since it cannot resolve the issue of key establishment without resorting to a third party.

Unlike symmetric-key cryptography system, the asymmetric-key cryptography system (more popularly known as public-key system) uses two keys, one for encryption and the other for decryption. The key used for encryption is the public key, which is accessible to the public and can be distributed widely and easily. The other one used for decryption is the private key, which must be kept secret and is only known to the owner of

the cryptosystem. By differentiating the encryption key and decryption key, the asymmetric-cryptography system can provide very important and unique security services such like key exchange and digital signature. A drawback of asymmetric-cryptography systems is that they have higher computational complexity, compared to symmetrical key systems.

Since Diffie and Hellman proposed the Diffie-Hellman key exchange scheme as the first asymmetric-cryptography system in 1976 [3], several asymmetric-key cryptography systems have been presented, such like RSA, ElGamel, and Elliptic curve cryptography. All these algorithms are based on some different hard mathematical problems. Based on their underlying mathematical problems, these algorithms can be classified as follows.

Public key system	**Hard Math Problems**
RSA	Integer factorization
ElGamal	Discrete logarithm problem
Elliptic curve cryptosystem	Elliptic curve discrete logarithm problem

Table 1.1 Public key systems and the hard math problems

The security strength of the cryptosystem relies on the fact it is hard to solve these mathematical problems. The Elliptic Curve Cryptosystem (ECC), first proposed by Miller and Koblitz in 1985 [1], [2], can provide higher security strength per bit compared to other asymmetric-cryptography system such like RSA. This is because elliptic curve discrete logarithm problem (ECDLP) is much harder to solve than factorization of a product of two large primes.

Security Level (bits)	RSA Key Size (bits)	ECC Key Size (bits)
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	512

Table 1.2 A comparison of key sizes

The large RSA key size requires long computation time and large VLSI area when implemented in hardware. While ECC is based on the elliptic curve discrete logarithm problem, the security of ECC relies on the difficulty of elliptic curve discrete logarithm problem. ECC can provide same level security strength as RSA with much shorter keys. Table 1.2 shows a comparison of key sizes needed to achieve equivalent level of security strength. This feature of ECC makes it very suitable for smart cards, credit cards, pagers, PDAs and mobile phones [4].

Other than solving the difficult ECDLP to break ECC mathematically, attackers can take advantage of the physical implementation and retrieve secret parameters by observing the information leaked during the computation. This method is referred to as Side Channel Attacks (SCA) and discussed in [5], [6]. Scalar multiplication, which multiplies a point on an elliptic curve by a scalar, is the main computation involved in ECC. Unguarded scalar multiplication algorithms are vulnerable to SCAs. In order to protect

ECC from SCA, the computation steps involved in the scalar multiplication algorithms have to be regular so that less useful side channel information can be leaked to attackers. There have been several existing algorithms for computing the scalar multiplication. A classical algorithm is the binary algorithm or doubling-and-add., it can be easily cracked by Simple Power Analysis (SPA) since it may consume different power in each iteration, depending on binary bit of the scalar. To withstand SPA, a double and add always algorithm is proposed by Coron in 1999 [7]. The idea is to add a dummy operation to make every iteration consume same power, thus the computation is regular in each step. However, this algorithm still remains insecure against a doubling attack proposed by Fouque in 2003 [8]. To provide further protection against various SCAs, Montgomery Powering Ladder (MPL) is invented by Marc Joye and Sung-Ming Yen in 2003 [9]. Although MPL is highly regular and efficient compared to the classical algorithms, it is still subject to SCAs such like Differential Power Analysis (DPA). Introduced by Kocher in 1999 [10], Differential Power Analysis focus on capturing the power consumption of the target device and by analyzing the power consumption to get information of the secret key. Another Modified Montgomery Power Ladder algorithm proposed by He in [11] can provide protection to more SCAs than the regular MPL. To the best of our knowledge, scalar multiplication based on this algorithm has not yet been implemented. Thus an efficient implementation of ECC, which can resist more SCAs is needed for modern cryptographic applications.

1.2 A Summary of Contributions

In this thesis we propose an efficient hardware architecture for the modified MPL with sequence masking (SM), exponent splitting (ES) and point randomization (PR)

algorithm [11]. More specifically we

- Proposed an efficient hardware architecture for the modified MPL with SM, ES and PR algorithm [11].
- Presented a FPGA implementation for the modified MPL with SM, ES and PR algorithm [11], which is the first time in literature.

1.3 Thesis Organization

The rest of the thesis is organized as follows. Chapter II provided mathematical background in finite field and elliptic curve which are important for understanding the proposed work. The concept of SCA is introduced and then many types of SCAs are reviewed in Chapter III. In Chapter IV an overview of existing related works is given. Chapter V provides a detailed discussion on a modified MPL with SM, ES and PR. Chapter VI proposes an efficient hardware architecture for the modified MPL with SM, ES and PR. FPGA implementation of the proposed architecture is also presented and the FPGA results are analyzed and discussed. Conclusive remarks are given and possible future works are commented in Chapter VII.

CHAPTER II

MATHEMATICAL BACKGROUND

This chapter introduces the related mathematical background of the ECC based systems. Definition of finite field and elliptic curve are given. Arithmetic over finite field, point operations over elliptic curve is also curved. Finally, some ECC scheme is introduced.

2.1 Finite Field

Finite field, also known as Galois Field (GF), is proposed by Galois in 1832. Galois Theory emphasis a relation between groups and fields. Finite field was introduced as an example of a field. A finite field is a finite set of numbers in which addition and multiplication are defined. It is an additive group under the addition operation. All the nonzero elements in a finite field form a multiplicative group under multiplication operation. Primitive element is the generator of the multiplicative group. For a finite field F, if n is the smallest integer satisfying that $na = 0$ for every field element a in the finite field F, then n is the characteristic of F.

Prime finite field $GF(p)$ consists of elements $\{0,1,2 \dots p-1\}$, where p is a prime number. Arithmetic in $GF(p)$ can be described as follows. Addition is defined as modulo-p addition. Multiplication is defined as modulo-p multiplication. p is the characteristic of $GF(p)$.

Extension finite field $GF(p^m)$, where p is a prime and m is a positive integer greater than 1, the elements of $GF(p^m)$ are polynomials of degree up to $m-1$ with coefficients belonging to $GF(p)$,

$$GF(p^m) = \{a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \cdots + a_2x^2 + a_1x + a_0\}$$

where $a_i \in GF(2)$, irreducible polynomial is $f(x)$. p is a prime and m is a positive integer

greater than 1, the elements of $GF(p^m)$ are polynomials of degree up to $m-1$ with coefficients belonging to $GF(p)$. The irreducible polynomial $f(x)$ cannot be factored into product of polynomials that has degree less than m.

Binary Extension finite field $GF(2^m)$ is a special case of $GF(p^m)$, where is consists of 2^m elements and its characteristic is 2.

$$GF(2^m) = \{a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \cdots + a_2x^2 + a_1x + a_0\},$$

where $a_i \in GF(2)$, irreducible polynomial is $f(x)$. Elements in this field can be represented as $(m-1)$ degree polynomial, for example if A is an element in $GF(2^m)$, A can be represented as

$$A = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \cdots + a_2x^2 + a_1x + a_0, where\ a_i \in GF(2)$$

It can also be represented as an m bits binary string $(a_{m-1}, a_{m-2}, ..., a_1, a_0)$, where $a_i \in GF(2)$. This representation is defined as the polynomial basis representation. This representation is beneficial in hardware implementation since its operations like addition and multiplication can be realized using AND logic gate and XOR logic gate. The addition operation over binary extension field is modulo 2 addition. The multiplication is modulo $f(x)$, modulo 2 multiplication.

2.2 Elliptic Curve over GF(2^m)

Let p be a prime number greater than 3, and coefficients a, b in the field $GF(p)$, then the elliptic curve E over $GF(p)$ is defined with equation:

$$\text{E: } y^2 = x^3 + ax^2 + b$$

This equation is a simplified Weierstrass equation. The points on the curve satisfy that both the x-coordinate and y-coordinate are both elements over $GF(p)$. Assume there

two points $P\ (x_1, y_1)$ and $Q\ (x_2, y_2)$ on the curve. The group operator point addition obeys the following rules. Draw a line through point P and Q, if there exists a third point $R\ (x_3, y_3)$ which intersects with the curve. The mirror reflection of R about the x-axis is defined as the addition result. If such an intersection point does not exist, we consider the result as infinity. The point at infinity O, defined by $P + O = O$, exists for every elliptic curve. The additive inverse of point P is its reflection across the x-axis. All the points on curve E and the point at infinity forms a group G defined by the point addition operator. G is an abelian group sine the group operator addition is commutative.

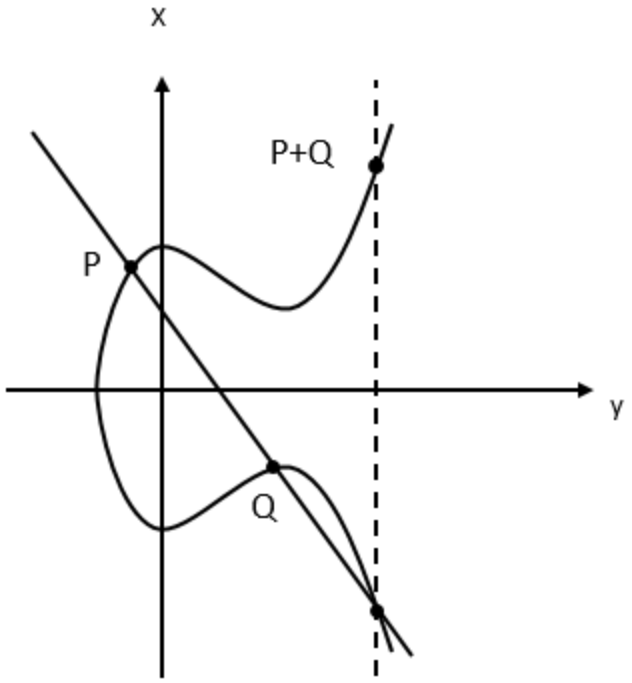

Figure 2.1 Point addition on ECC

If P and Q satisfy $x_1 \neq x_2$, then we have point addition computed using the following equation.

$$\begin{cases} \lambda = \dfrac{y_1 + y_2}{x_1 + x_2} \\ x_3 = \lambda^2 + x_1 + x_2 \\ y_3 = (x_1 + x_3)\lambda + y_1 \end{cases}$$

If P and Q satisfy $P = Q$ and $y_1 \neq 0$, then point doubling is defined below.

$$\begin{cases} \lambda = \dfrac{3x_1^2 + a}{2y_1} \\ x_3 = \lambda^2 + 2x_1 \\ y_3 = (x_1 + x_3)\lambda + y_1 \end{cases}$$

For an integer k, scalar multiplication kP is defined as repeated addition like

$$kP = P + P + \cdots P$$

repeated for k times.

The order n of point P is defined as the minimal integer satisfies $nP = O$.

Let a and b be elements in $GF(2^m)$. An elliptic curve E over $GF(2^m)$ can be defined by the equation

$$\text{E: } y^2 + xy = x^3 + ax^2 + b$$

Points over elliptic curve defined in $GF(2^m)$ have similar operations to that defined in $GF(p)$. Consider two points $P\ (x_1, y_1)$ and $Q\ (x_2, y_2)$ on the curve, and $R\ (x_3, y_3)$ be the result of $P + Q$. If $P = Q$, point addition is computed as follows.

$$\begin{cases} \lambda = \dfrac{y_1 + y_2}{x_1 + x_2} \\ x_3 = \lambda^2 + \lambda + x_1 + x_2 + a \\ y_3 = (x_1 + x_3)\lambda + x_3 + y_1 \end{cases}$$

If $P \neq Q$, point doubling is performed as below.

$$\begin{cases} \lambda = \dfrac{y_1}{x_1} + x_1 \\ x_3 = \lambda^2 + \lambda + a \\ y_3 = {x_1}^2 + \lambda x_3 + x_3 \end{cases}$$

All the addition, multiplication and inversion involved in there equations are finite field arithmetic. The most time consuming operations are finite field multiplication and inversion.

2.3 Elliptic Curve

For a given field $GF(p)$ and curve E over the field, it is easy to calculate kP for a point P on the curve. However, if kP and P is known, it is considered very difficult to calculate integer k. This is called the ECDLP problem, and the security strength of ECC relies on it.

2.3.1 System Setup

To setup a secure elliptic curve cryptography system, the parameters of the curve has to be chosen very carefully. The elliptic curve used in the system need to be non-singular since singular curves are easy to crack.

The parameters needed to construct an elliptic curve cryptography system including the following. Finite field $GF(p)$ defined by p, elliptic curve E defined by a and b, base point P, the order of the base point n and the factor h. The parameter set is known to the public.

If the system is over $GF(p)$, the parameter set is $D = \{p, a, b, P, n, h\}$. The parameters are defined as above stated. If the cryptography system is built over $GF(2^m)$, the parameter set is $D = \{m, f(x), a, b, P, n, h\}$ where $f(x)$ is the irreducible polynomial define the field. To calculate factor h, first count all the points and the point at infinity to get the order of the elliptic curve #E, then h is #E divided by n.

2.3.2 Generation of the key pairs

After the system is set up, assume Bob is trying to communicate with Alice. Alice

will choose the keys as introduced below.

Step 1: Choose a random integer $d \in [1, n-1]$

Step 2: Computes point $Q = dP$

The public key is point Q, and private key is integer d.

2.3.3 Elliptic Curve Integrated Encryption Scheme (ECIES)

The most extended encryption and decryption scheme based on ECC is the Elliptic Curve Integrated Encryption Scheme (ECIES). This scheme is a variant of the ElGamal scheme proposed in [12]. This scheme is described as follows:

System Setup:

Step 1: Alice sets the ECC system parameter sets $D = \{p, a, b, P, n, h\}$

Step 2: Computes Key Pair (Q, d)

Encryption:

Step 1: Bob selects a random number $k \in [1, n-1]$

Step 2: Computes $R = kP$, and $Z = hkQ$.

Step 3: X-coordinate of Z and R is converted to (k_1, k_2) using a key derivation hash function

Step 4: Message m is encrypted with k_1 using a symmetrical key cipher to get $C = ENC\ (m)$

Step 5: Computes $t = MAC\ (C)$ using k_2, where MAC is a message authentication code

Step 6: Cipher text (R, C, t) is sent to Alice

Decryption:

Step 1: Alice computes $Z = hdR$

Step 2: X-coordinate of Z and R is converted to (k_1, k_2) using a key derivation hash

function

Step 3: Computes $t = MAC\ (C)$ using k_2, where MAC is a message authentication code

Step 4: Message m is decrypted with k_1 using a symmetrical key cipher to get $m = DEC\ (C)$

This scheme works since when Alice generates Z, it follows that,

$$Z = hdR = hd\ (kP) = hk(dP) = hkQ$$

So both the encryption and decryption generate the same key pair (k_1, k_2).

2.3.4 Elliptic Curve Diffie–Hellman Key Exchange (ECDHKE)

ECDHKE is the generic key exchange scheme based on the Diffie-Hellman mechanism applied to elliptic curves [13]. The information available to the public is the elliptic curve E over $GF(p)$, and a point P with order n on the curve. The computations are described as follows:

Computation of Alice:

Step 1: Alice chooses random number $a \in [1, n-1]$

Step 2: Computes aP and it is sent to Bob

Step 3: After receiving bP form Bob, Alice computes scalar multiplication $a(bP)$ to get abP

Computation of Bob:

Step 1: Bob chooses random number $b \in [1, n-1]$

Step 2: Computes bP and it is sent to Alice

Step 3: After receiving aP form Alice, Bob computes scalar multiplication $b\ (aP)$ to get baP

The secret key shared by the two parties is $k = abP$. By applying this scheme, the

secret key is securely exchanged.

2.3.5 Elliptic Curve Digital Signature Algorithm (ECDSA)

ECDSA is the elliptic curve variant of the Digital Signature Algorithm [14]. This scheme can be introduced as the following steps:

System Setup:

Step 1: Alice sets the ECC system parameter sets $D = \{p, a, b, P, n, h\}$

Step 2: Computes Key Pair (Q, d)

Signing of the message:

Step 1: Alice calculates $e = H(m)$, where H is a hash function.

Step 2: Chooses random number $k \in [1, n-1]$

Step 3: Calculates point $R = kP\ (x_1, y_1)$.

Step 4: Calculates $sig_1 = x_1 \bmod n$. If $sig_1 = 0$, Alice chooses another k.

Step 5: Calculates $sig_2 = k^{-1}(e + dx_1) \bmod n$.

Step 6: The signature is the pair (sig_1, sig_2).

Verification of the signature:

Step 1: Bob calculates $e = H(m)$, where H is a hash function.

Step 2: Calculates $w = sig_2^{-1} \bmod n$.

Step 3: Calculates $u_1 = ew \bmod n$ and $u_2 = sig_1 w \bmod n$.

Step 4: Calculates point $X = u_1 P + u_2 Q$ and retrieve the x-coordinate of X denoted by x_2

Step 5: The signature is valid if $sig_1 = x_2 \bmod n$, if not the signature is rejected.

To prove the correctness of the algorithm, the following equation can be performed.

$X = u_1 P + u_2 Q$

$= u_1 P + u_2 d \times P$

$= (u_1 + u_2 \mathrm{d}) \times \mathrm{P}$

$= (e \times sig_2^{-1} + sig_1 \times sig_2^{-1} d) \times P$

$= (e + sig_1 d) sig_2^{-1} \times P$

$= (e + sig_1 d)\,(e + sig_1 d)^{-1} (k^{-1})^{-1} \times P = kP$

Thus from the computation it is verified $sig_1 = x_2$ mod n from point $X = kP$.

From all the above algorithms, it can be seen that scalar multiplication is the main computation involved in elliptic curve cryptography. SCAs mainly target the scalar multiplication in ECC to compromise partial or the full secret key.

CHAPTER III

SIDE CHANNEL ATTACKS

To consider a cryptosystem under mathematically circumstances, a black-box model is usually used. In the black-box scheme, attackers cannot get any intermediate computation results [15]. The only information available to the attackers are plaintext and cipher text. Thus in order to break the cryptosystem, the attackers need to solve the hard math problem such like ECDLP in ECC.

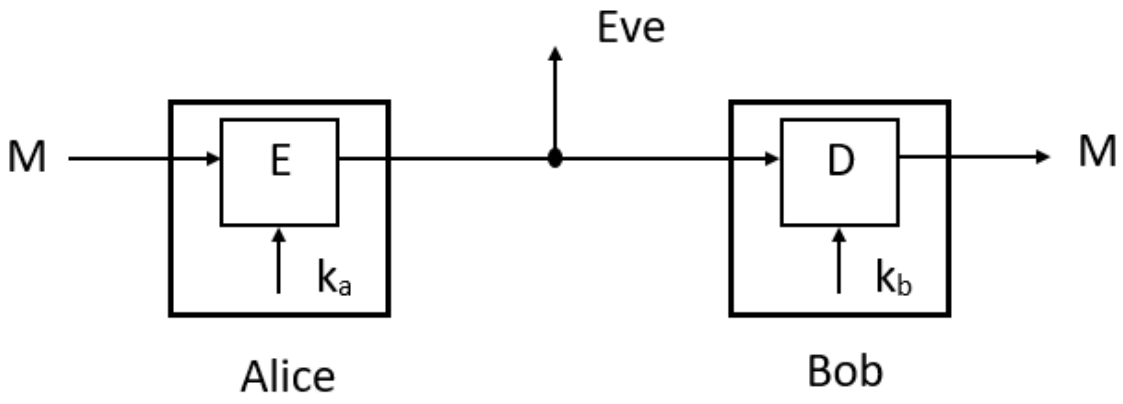

Figure 3.1 Traditional cryptosystem

However, such model is not adequate under most scenarios in practice. When a cryptosystem is implemented on hardware, there will be unintended information leaked during the execution of the algorithm. The attackers thus can try to find correlation between the leaked information and the secret key. Side Channel can be classified based on the types of side channel information. Information can be extracted from timing, power consumption or electromagnetic radiation features. Hardware or software faults, computational errors, and changes in frequency or temperature can also lead to leak of information. Table 3.1 shows the relation between leaked information and type of side channel attacks.

Side Channel Information	Side Channel Attacks
Power traces	Simple power analysis(SPA)
	Differential power analysis
	Comparative power analysis
Time	Timing attack
Faults and error	Fault attack
Electromagnetic radiation	EM attack

Table 3.1 Category of SCAs

3.1 Timing Attack

Many of the implementations of cryptographic algorithms perform the computations in a non-constant time. The time variations sometimes can become a breakthrough point of the system. If the difference of time is correlated to the secret parameters, then a statistical analysis can reveal enough information to access the key. Timing attack was first introduced by Kocher in [5].

The working principle of timing attack is to capitalize on the time difference, thus an easy countermeasure is to make the implementation on the hardware equalize the computation time in each step.

3.2 Fault Attack

The idea of fault attack is to inject a fault and force the system to leak information related to the secret parameter. Fault attack was first introduced in [16]. Consider a hardware implementation of a cryptosystem, the execution of a fault attack usually two steps [17]. The first step is to inject a fault and the second step is to do a cryptanalysis based on the erroneously result.

Among the many fault attacks, differential fault attack (DFA) caught most attention of the scientists and researchers. To initialize DFA, a bit error is enforced into the hardware before or during the computation. By analyzing the correct result and the erroneous result, the information of the secret key is compromised. Biehl and Műller proposed an effective DFA against ECC in 2000 [18]. The main idea is to construct a reference elliptic curve E_r and choose a reference point P_r on the Curve. By inputting this reference point into the tamper-proof hardware, the reference result $d \times P_r$ on the curve is E_r computed. Since curve E_r is carefully chosen that the order of the curve has a factor r equals to the order of P_r, the ECDLP is deduced to the subgroup of order r. Next another reference point is carefully chosen and the whole process is duplicated. The secret parameter d can be revealed.

Yen and Joye introduced another fault attack scheme in [19], this attack targets the algorithms with dummy operations. By carefully timing a fault injection, the attacker can distinguish whether this fault causes an error in the result. Thus it can be known whether a redundant operation is executed. By repeating the procedure, the whole secret key can be retrieved. This attack is called safe-error attack.

3.3 Electromagnetic Attack

Electromagnetic (EM) attack [20] is based on the fact that all electrical device radiates electromagnetic waves when operating. An adversary examines the changes of the electromagnetic field first and then an electromagnetic analysis is executed trying to extract the secret information.

Wu and Yu introduced an EM attack against scalar multiplication in [21]. The attacker is targeting a classical binary algorithm. The electromagnetic traces captured have

significant difference when different computation is executed. Thus it is easy to access the secret key by comparing the power traces.

3.4 Power Analysis Attack

Similar to EM attacks, power consumption is often leaked during the running process of the cryptographic device. By observing the power traces, attackers can get access to the information on where a certain operation happened and what secret parameter is involved. Since power analysis attacks usually use little resources, power constrained applications such like smart cards are primary targets of it [22].

Power analysis attack can be generally categorized into two types, simple power analysis (SPA) and differential power analysis (DPA). SPA attacker examines the power traces of cryptography computations and distinguish the power consumption caused by the secret key. The attacker tries to get secret information by observation of repetitive patterns in the obtained power traces. If power consumptions are distinguishable the information of secret key is leaked. DPA relies on the statistical analysis to reveal the correlation between the secret bit and power consumption. It is usually executed in two steps. First the power traces are captured, then analysis regarding the captured information is carried out. Among all the various SCAs, DPA is being regarded as one of the strongest. The difference lies in that despite merely observation of the power spectrum, DPAs employ more statistical methods to guess the secret key.

Coron shows in [7] how DPAs work against scalar multiplication, which is the main computation in ECC. Since in many scalar multiplication algorithms, the result is computed by continuous point addition chain. That is, given a point P and scalar d, dP is calculated as:

$$P \to d_0 P \to d_1 P \to d_2 P \to d_3 P \to \cdots \to dP$$

The attacker starts guessing from $d_0 = 1$ to d. For each d_i, a set A_i with all possible $d_i = d_j + d_k$ where $0 \leq j \leq k < i$ is built. For each element d_i in the set A_i, the correlation of power consumption and $d_i P$ is computed. When the peak is achieved in the computation, the desired d_i is recovered. By repeating the same pattern for $d_0 = 1$ to d, the secret key d will be compromised at last.

CHAPTER IV

EXISTING WORK REVIEW AND SECURITY ANALYSIS

There have been different scalar multiplication algorithms existing to improve the overall performance for efficient or security. In this chapter we introduce the different algorithms and their implementations. Also the security strength of these algorithms will be analyzed.

4.1 Classical binary algorithm

The binary algorithm is also known as the "square and multiply method" to do exponentiation calculation. It is a very old algorithm which has over 2000 year's history [23]. While in the scalar multiplication scenario, it is very simply adapted as the double and add algorithm.

The basic idea of double and add algorithm is to make full use of the binary form of the scalar. Consider a point P and scalar k, rather than add point P to itself d times to get the result kP, the double and add method will obviously reduce number of addition operation needed.

The binary algorithm has two versions, one is from the least significant bit to the most significant bit shown in Algorithm 4.1.

Algorithm 4.1.Right to left version of double-and-add method

Input: $P \epsilon E, k = (k_{n-1}, \dots, k_1, k_0)_2 , k_i \epsilon$ 0,1

Output: $Q = kP \epsilon E$

Step 1: $x = O, y = P;$

Step 2: $for\ i{=}0\ to\ n{-}1\ do$

Step 3: $if\ k_i = 1, then\ x = x + y$

Step 4: $end\ if$

Step 5: $y = 2y$

Step 6: $end\ for$

Step 7: $Return\ x$

The other is calculating from the most significant bit to the least significant bit, as shown in algorithm 4.2.

Algorithm 4.2.Left to right version of double-and-add method

Input: $P \epsilon E, k = (k_{n-1}, \dots, k_1, k_0)_2\ , k_i \epsilon\ 0,1$

Output: $Q = kP \epsilon E$

Step 1: $x = P;$

Step 2: $for\ i{=}n{-}2\ down\ to\ 0\ do$

Step 3: $x = 2x$

Step 4: $if\ k_i = 1, then\ x = x + P$

Step 5: $end\ if$

Step 6: $end\ for$

Step 7: $Return\ x$

Although the addition operation is significantly reduced in the double-and-add algorithm, the drawbacks of this algorithm is crucial. Comparing the operations carried out when the secret bit k_i is different, it is clear that more computations are executed when

$k_i = 1$. In every iteration, the doubling operation is always performed, while addition only happens when $k_i = 1$. This variation responses directly into the power consumption. As shown in Figure 4.1, D denotes doubling operation and A represents addition operation. The secret bit is chosen as $k = (0101)_2$. The length of the power peak is different, thus the attacker can observe the power traces and distinguish the secret bits. This figure shows exactly how the double-and-add algorithm is vulnerable to the SPA attack. Because of this major drawback, many other algorithms is have been developed.

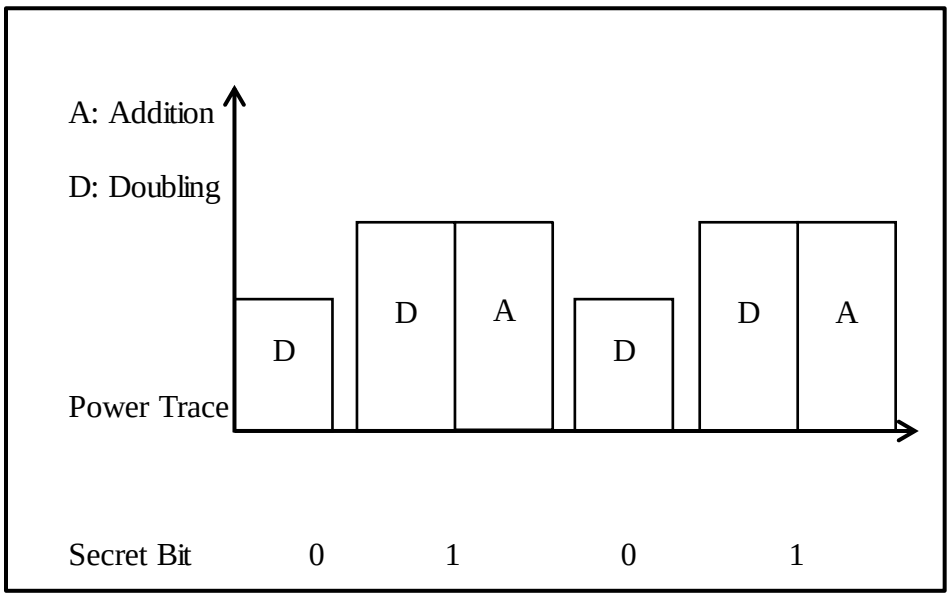

Figure 4.1 Vulnerability of binary method to SPA

A high speed ECC processor is implemented by Hossain and Kong in 2015 [24]. Over field $GF(2^{233})$, their ECC processor can perform scalar multiplication 2.66 ms at 255.66 MHz in on a Xilinx Kintex-7 devices. This design features to speed up the overall computation time. They applies the double-and-add algorithm to the design in a trade-off for speed. As stated previously, this design can be easily crack by SCA attacks.

4.2 Double-and-Add Always Algorithm

In order to achieve more security strength against SPA, Coron proposed a double-and-add always algorithm in 1999 [7]. The main idea of this algorithm is to add a dummy operation to standard double-and-add to equalize the power consumption of each cycle. As illustrated in Algorithm 4.3, a dummy addition is executed when the processing bit is zero. In this way, the algorithm is performing a doubling followed by an addition in every iteration. Since the algorithm is regular now, it has resistance to SPA attacks.

Algorithm 4.3.Double-and-add always method

Input: $P \epsilon E, k = (k_{n-1}, \dots, k_1, k_0)_2 , k_i \epsilon$ 0,1

Output: $Q = kP \epsilon E$

Step 1: $x = O, y = O;$

Step 2: $for\ i{=}n{-}1\ down\ to\ 0\ do$

Step 3: $x = 2x;$

Step 4: $y = x + P;$

Step 5: $if\ k_i = 1, then\ x = y;$

Step 6: $end\ if$

Step 7: $end\ for$

Step 8: $Return\ x$

However, another fault attack called c-safe error attack [9] can easily crack this algorithm. When the secret bit is zero, the addition result won't affect the final result. When the attacker altered one bit of the secret key to zero, the result can be compared with the original output. If the result remains the same, the attacker knows that the secret bit is zero.

In the case the result changed, the attacker will figure that the secret bit is one and it is quite easy to locate. This algorithm is taking as an unsuccessful sequence mask example since the countermeasure takes to prevent SPA benefits other attacks [26].

Another attack threatening this algorithm is the doubling attack, which proposed by Fouque and Valette in [25]. The name is from the fact this attack is based on the doubling operation in the scalar multiplication. This type of attack only works when using left to right algorithms. To better explain it, consider two points $P, 2P$ and secret $k = 9 = (1001)_2$ as input. Double-and-add always method is used to compute kP and $k\ (2P)$. As illustrated in Table 4.1, if we focus the doubling operation in each iteration, some of them shares the same pattern. To be more specific, doubling operation at iteration 2 calculating $k\ (2P)$ is the same as doubling operation at iteration 3 calculating kP. Thus two zeros is overserved by the adversary. This attack reveals all the bits with the value zero and using these information, the whole secret bits can be revealed consequently.

Table 4.1 Doubling attack against double-and-add always algorithm

Iteration i	k_i	kP	$k(2P)$
3	1	2×0 $0 + P$	2×0 $0 + 2P$
2	0	$2 \times P$ $2P + P$	$2 \times 2P$ $4P + 2P$
1	0	$2 \times 2P$ $4P + P$	$2 \times 4P$ $8P + 2P$
0	1	$2 \times 4P$ $8P + P$	$2 \times 8P$ $16P + 2P$

To the best knowledge, implementation for double-and-add always algorithm is very few since it is not efficiency compared to classical double-and-add. Although it has

more resistance to SPA, it is still vulnerable to many other attacks such like safe error attack. This method later stimulated development of more advanced algorithms such like MPL.

4.3 Non Adjacent Form Method

As stated in previous chapter, a point $P\ (x,y)$ on elliptic curve E over binary field, has additive inverse in the form $-P\ =\ (x, x+y)$. Thus subtraction of point is actually the same as point addition on an elliptic curve. The non-adjacent form (NAF) is a signed digit representation [27] inspired by this fact. In NAF method, the secret key is represented by the following equation.

$$k = \sum_{i=0}^{l-1} k_i 2^i, where\ k_i \in \{0, \pm 1\}, k_{l-1} \neq 0$$

In this expression, there is no two continuous nonzero digits. The advantage of this representation is that generally it has fewer nonzero bits [23], which leads to a reduction of addition operation needed in the algorithm. The NAF method is shown in Algorithm 4.4.

Algorithm 4.4. Non Adjacent Form method

Input: $P \epsilon E, NAF(k) = \sum_{i=0}^{l-1} k_i 2^i, k_i \epsilon \{0, \pm 1\}$

Output: $Q = kP \epsilon E$

Step 1: $x = P, y = O;$

Step 2: $for\ i{=}l{-}2\ down\ to\ 0\ do$

Step 3: $x = 2x;$

Step 4: $if\ k_i = 1, then\ x = x + P;$

Step 5: else $if\ k_i = -1, then\ x = x + (-P);$

Step 6: $end\ if$

Step 7: $end\ for$

Step 8: $Return\ x$

The doubling attack mentioned in previous section still applies to the NAF method. Taking an integer $k = 29$, and $NAF\ (k) = (1, 0, 0 - 1, 0, 1)$. Then if the adversary calculates kP and takes $k\ (2P)$ as a reference. From Table. 4.2, it is shown that the intermediate value at iteration i when computing $k\ (2P)$ equals the result at iteration i-1 when computing kP. All the secret bits with value zero is thus revealed. Since approximately two third of the bits in NAF representation is zero [28], most bits are retrieved applying doubling attack.

Table 4.2 Doubling attack against NAF method

Iteration i	k_i	kP	$k(2P)$
4	0	$2 \times P$	$2 \times 2P$
3	0	$2 \times 2P$	$2 \times 4P$
2	-1	$2 \times 4P$ $8P + (-P)$	$2 \times 8P$ $16P + (-2P)$
1	0	$2 \times 7P$	$2 \times 14P$
0	1	$2 \times 14P$ $28P + P$	$2 \times 28P$ $56P + 2P$

Wang introduced his implementation of an ECC coprocessor over $GF(2^{233})$ in 2005 [29]. In this paper, a digit-serial multiplier which can achieve half or quarter clock cycles compared to the full-serial multiplier is proposed. With proper precomputation, this design can reduce the total calculating time by applying the projective coordinates. The coprocessor can perform a scalar multiplication in 2.28 ms at 80 MHz. However, the implementation algorithm chosen by the author is NAF method which vulnerable to attacks like doubling attack.

4.4 Montgomery Powering Ladder Algorithm

4.4.1 Explanation of Algorithm

The Montgomery powering ladder is first proposed in [9] to provide a countermeasure to SPA. For a given $k = \sum_{i=0}^{l-1} k_i 2^i$, in order to compute kP, the MPL is constructed based on the following relationships. Define $L_j = \sum_{i=j}^{l-1} k_i 2^i$ and $H_j = L_j + 1$. L_j and $H_j$can then be represented as

$$\begin{cases} L_j = 2L_{j+1} + k_j = L_{j+1} + H_{j+1} + k_j - 1. \\ H_j = L_{j+1} + H_{j+1} + k_j \end{cases}$$

So at iteration j, it is easy to express L_j and H_j using the previous values from iteration $j + 1$

$$(L_j, H_j) = \begin{cases} (2L_{j+1}, L_{j+1} + H_{j+1}) \; when \; k_j = 0 \\ (L_{j+1} + H_{j+1}, 2H_{j+1}) \; when \; k_j = 1 \end{cases}$$

The above equations implies the structure of MPL. The calculations are very similar to each other and in each iteration one doubling is performed with one addition. Recall that the main computation of scalar multiplication is point doubling and point addition, the following algorithm is obtained.

Algorithm 4.5. Montgomery Powering Ladder

Input: $P \epsilon E, k = (k_{n-1}, \dots, k_1, k_0)_2 , k_i \epsilon$ 0,1

Output: $Q = kP \epsilon E$

Step 1: $x = 0, y = P;$

Step 2: $for\ i{=}n{-}1\ down\ to\ 0\ do$

Step 3: $if\ k_i = 0, then\ y = x + y;\ x = 2x$

Step 4: $else\ if\ k_i = 1, then\ x = x + y; y = 2y$

Step 5: $end\ if$

Step 6: $end\ for$

Step 7: $Return\ x$

4.4.2 Advantage of MPL

In algorithm 4.5, it is obvious that the computations for point addition and point doubling are independent. Thus they can be calculated parallel. This feature results in faster calculation speed. Moreover, since in every circle, MPL executes the same operation, it is considered highly regular. This structure makes it invulnerable to SPA.

Another c safe-error attack introduced in previous section also has no effect to MPL. Unlike the double-and-add always algorithm, there is no dummy operation in MPL. So any fault injected will lead an error in the result.

The doubling attack is considered ineffective. Assume there are two registers R_0 and R_1 to store the intermediate values. It could be the value stored in R_0 or R_1 performs the doubling, depending on the secret bit. While the previous algorithms which are

vulnerable to doubling attack all shares the fixed doubling pattern, the attacker cannot see a repeated doubling operation when there is no continuous bits with the same value one or zero. But if there exists successive bits carrying the same value, the algorithm is vulnerable to a relative doubling attack proposed in [30].

4.4.3 Relative Doubling Attack against MPL

While the doubling attack mainly focus on finding the 0 bits, the relative doubling attack compares the value of two adjacent secret key bits. Let registers R_0 store the value of L_i, and registers R_1 store the value of H_i. If $k_i = k_{i-1} = 0$, the following doubling computation is observed.

$$\begin{cases} R_0 \leftarrow 2 \times L_i P\text{: } iteration\ i-1\ when\ calculating\ kP \\ R_0 \leftarrow L_{i+1} \times 2P\text{: } iteration\ i\ when\ calculating\ k(2P) \end{cases}$$

Since $L_i = 2L_{i+1}$ when obviously the above equations are doing the same computation. Thus, if such collisions are observed, the attacker get the information $k_i = k_{i-1} = 0$.

Similar situation applies to when $k_i = k_{i-1} = 1$. Then in register R_1,

$$\begin{cases} R_1 \leftarrow 2 \times H_i P\text{: } iteration\ i-1\ when\ calculating\ kP \\ R_1 \leftarrow H_{i+1} \times 2P\text{: } iteration\ i\ when\ calculating\ k(2P) \end{cases}$$

same computations will be carried because that $H_i = 2H_{i+1}$. While when ki is not equal to k_{i-1}, there is no collision detected.

Taking $k = 105 = (1101001)_2$, the adversary computes kP and $k(2P)$ as the pattern shown in Table. 4.3. It is very clear that at iteration 1 and 2, collisions in register R_0 is detected. This fact leads to secret key bits $k_2 = k_1 = 0$. Taking iteration 5 and 6 as another example, the operations regarding values in register R_1 is the same, thus the attacker figures $k_6 = k_5 = 0$. When there is no collision happened, that means the two

adjacent secret bits are holding different values. Then the whole secret key is revealed from the collision location bit by bit.

Table 4.3 Relative doubling attack against MPL

Iteration i	k_i	kP	$k(2P)$
6	1	$R_0 = 0 + P$ $R_1 = 2 \times P$	$R_0 = 0 + 2P$ $R_1 = 2 \times 2P$
5	1	$R_0 = P + 2P$ $R_1 = 2 \times 2P$	$R_0 = 2P + 4P$ $R_1 = 2 \times 4P$
4	0	$R_0 = 2 \times 3P$ $R_1 = 3P + 4P$	$R_0 = 2 \times 6P$ $R_1 = 6P + 8P$
3	1	$R_0 = 6P + 7P$ $R_1 = 2 \times 7P$	$R_0 = 12P + 14P$ $R_1 = 2 \times 14P$
2	0	$R_0 = 2 \times 13P$ $R_1 = 13P + 14P$	$R_0 = 2 \times 26P$ $R_1 = 26P + 28P$
1	0	$R_0 = 2 \times 26P$ $R_1 = 26P + 27P$	$R_0 = 2 \times 52P$ $R_1 = 52P + 54P$
0	1	$R_0 = 52P + 53P$ $R_1 = 2 \times 53P$	$R_0 = 104P + 106P$ $R_1 = 2 \times 106P$

4.4.4 M-safe Error Attack against MPL

Despite the relative doubling attack, another fault attack M safe-error attack introduced in [19] is also proven to be effective against MPL. In this scheme, the fault induced by the attacker is temporary memory fault and it is very carefully timed. M safe-error attack takes advantage of the two distinct operations carried out in each cycle of MPL. Consider the different computation when k_i is different. When $k_i = 1$, the two operations are $R_0 = R_0 + R_1$ and $R_1 = 2R_1$. Any fault injected into R_1 will change the value of the result, thus it is not a safe error. If $k_i = 0$, then $R_0 = 2R_0$ and $R_1 = R_0 + R_1$ will be computed, error inputted into more significant bits of R_1 will be erased after the

resigning of value in R_1. In other words, the result still remains and the error is regarded a safe-error. The attacker can retrieve the secret bit k_i based on the above fact.

4.4.5 Comparative Power Analysis against MPL

Comparative Power Analysis is proposed by Homma in 2010 [31]. It is a more powerful attack which can compromise multiple scalar multiplication algorithms such like double-and-add algorithm, double-and-add always algorithm and MPL algorithm. While the original attack introduced is aiming at implementation of exponentiation, it also applies to scalar multiplication.

Similar to the relative doubling attack, the basic idea of comparative power analysis is also generating collisions by inputting a pair of carefully chosen message. More specifically, the chosen inputs are set to satisfy the equation $\alpha P = \beta Q$, where α and β are integer. The input P is computed with the target operation αP while the input Q is illustrates the reference power trace. When the collision happens, the doubling operation of the two input will have very similar power trace. By comparison of the power traces, the target secret key bit is leaked. Unlike doubling attack, the collision can be retrieved from different time frames.

Figure 4.2 shows an example how the attack scheme works. The input condition is chosen as $15P = 2Q$. Consider the first four bits of the secret key is known to the attacker, the target bit is the fifth bit. The attacker then make an assumption that the target bit is 1, and illustrates the power traces. Then he examines the reference power traces at the time frame $2Q$ doubling is computing, if the power trace shares similarity with the target power trace at the time frame computing $15P$ doubling, the target secret bit is 1. The assumption made at the beginning is correct. If there is no similarity detected in the target and reference

power trace, the secret bit is 0. Note that the collision is generated at different timeframe of the two power traces. By repeating the attack pattern, all the bits of the secret key will be known.

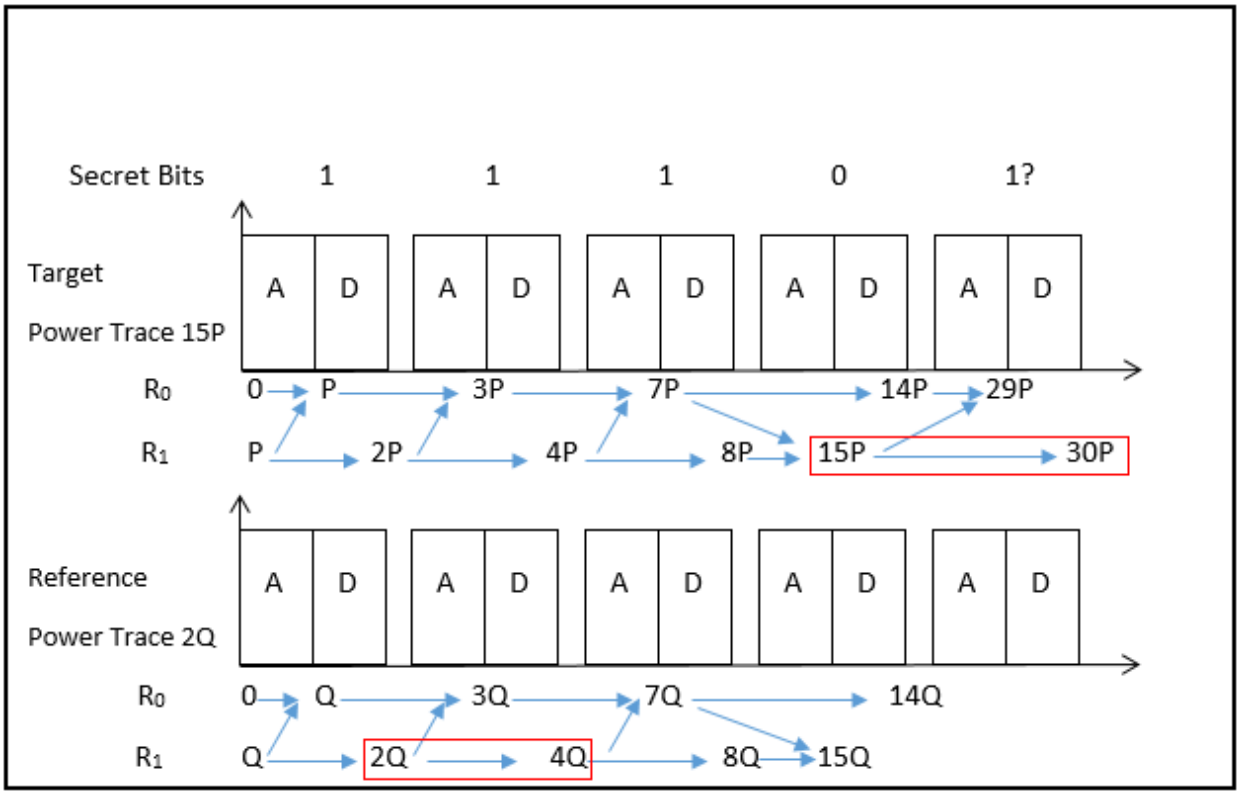

Figure 4.2 Comparative power analysis against MPL

4.4.6 MPL based hardware implementation

Deschamps and G.Sutter implemented EC scalar multiplication over $GF(2^{163})$ in 2008 [32]. This design is implemented using MPL algorithm. The computation time is 1ms at 100MHz and a comparison with binary algorithm implementation has been made. As stated previously, implementation on unprotected MPL is still vulnerable to some SCAs, thus an implementation on a more secure algorithm is proposed in the next chapter.

CHAPTER V

ANALYSIS OF MODIFIED MPL WITH COUNTERMEASURES

In this chapter we introduce a modified MPL with sequence masking, exponent splitting and point randomization proposed in [11]. A small modification has been done to this algorithm to make it suitable for ECC scalar multiplication since it is originally invented for exponentiation operation.

5.1 Existing countermeasure techniques

In previous Chapters, it has been stated that unprotected MPL is still vulnerable to a lot of side channel attacks. To offer protections in algorithm level, He, Huang and Wu proposed a highly secure MPL for exponentiation operation [11]. Several countermeasures had been applied to enhance its security strength.

5.1.1 Coron's three countermeasures to DPA

Coron has proposed three countermeasures against DPA to improve the original MPL [7]. The first is to randomize the secret private exponent. The main idea is to change the representation of the secret exponent. Denote $\#\varepsilon$ as the number of all the points on the curve E, and select a random number k. Then the following computation is carried out to calculate $d' = d + k\#\varepsilon$, the computation Q = dP is replaced by $Q = d'P$. The correctness of this transform is based on the fact that $\#\varepsilon P = 0$. To expand this equation in detail, we get the following proof.

$$Q = d'P = (d + k.\#\varepsilon)P = dP + k.\#\varepsilon.P = dP + O = dP$$

The size of the random number k is suggested to be 20 for better security performance. By applying this countermeasure, the compute process is changed while the result Q still remains the same.

The second countermeasure is to blind the point P. Choose a random R on the same curve as point P. Point R is used as a mask to provide protection to the point P. Another point S is computed as $S = dR$. The cryptographic system then performs the scalar multiplication $d(R + P)$. In order to retrieve $Q = dP$, a subtraction will be applied the previous result. Since S is already known, the final result will be recovered by $d(R + P) - S$. Point R and S will be updated in every initialization of the scalar multiplication.

The third countermeasure is to randomize to projective representation [33] of the point $P\ (x, y)$. Point P can be represented in projective coordinates as (X, Y, Z). Since it is not the only projective representation point P has, it can be written as $(\lambda X, \lambda Y, \lambda Z)$ where $\lambda \neq 0$. The binary representation of point P is then guarded. Since this thesis mainly focus on implementation in affine coordinates, this countermeasure will not be discussed in detail.

5.1.2 Exponent Splitting

Exponent Splitting technique is first proposed in [34]. While it is used in exponentiation is the paper, the idea is the same in scalar multiplication. The basic thought is based on the simple observation that

$$dP = (d - a)P + aP$$

In this splitting technique, the scalar d can be split into two parts as the above equation indicated. First a random number r is generated, where r is smaller than d. r' is then calculated by $r' = d - r$. The scalar multiplication then become

$$dP = rP + r'P$$

since $d = r + r'$, it is obvious that the result is the same without scalar splitting.

Since we can see from the equation, the splitting technique is basically transforming a single scalar multiplication into two separate computations, thus it will be more time

consuming to put this technique into practice. Usually it takes two times the original time. But considering the security strength, sometimes the trade-off is worthy. This technique offers enhanced protection against SPA, and the randomization also helps to prevent some differential attacks.

5.1.3 Blinded Fault Resistant Exponentiation

This technique is first proposed by Fumaroli and Vigilant in [35]. It is an extension of Coron's second countermeasure [7]. The main idea is to add a mask to the base point P to construct a masked MPL. The algorithm is originally for exponentiation operation, but by a small modification, it can be also applied to scalar multiplication.

Algorithm 5.1. Masked Montgomery Powering Ladder

Input: $P \epsilon E, k = (k_{n-1}, \dots, k_1, k_0)_2 , k_i \epsilon$ 0,1

Output: $Q = kP \epsilon E$

Step 1: $Select\ random\ point\ R\ on\ E;$

Step 2: $R_0 = R, R_1 = P + R, R_2 = -R$

Step 3: $for\ i{=}n{-}1\ down\ to\ 0\ do$

Step 4: $if\ k_i = 0, then\ R_1 = R_0 + R_1;\ R_0 = 2R_0; R_2 = 2R_2$

Step 5: $else\ if\ k_i = 1, then\ R_0 = R_0 + R_1; R_1 = 2R_1;\ R_2 = 2R_2$

Step 6: $end\ if$

Step 7: $end\ for$

Step 8: $Return\ Q = R_0 + R_2$

In Algorithm 5.1, at the beginning of the operation, a random point R is picked. R is acting as a mask to the base point P. Three registers R_0, R_1, R_2 are needed for the

algorithm. R_0 is initialised with the mask R, R_1 is initialised with $P + R$ and R_2 is given the anti-mask $-R$. The mask is updated at each iteration. The intimidate values in R_0 and R_1 are blinded with the updated mask $2^{n-i}R$. The anti-mask value is also updated at every iteration. The value in register R_2 is following the pattern $2^{n-i}(-R)$. At the end of the computation, the final unmask step will be executed to recover the original desired result.

This algorithm can provide resistance to more attacks while still keeps the feature that regular MPL offers. Since the regular MPL is highly regular, the algorithm is insensitive to SPA. As the mask technique is applied in this algorithm, the intermediate values are independent from the input and output according to [35]. Thus the attacker cannot exploited the secret key by DPA. Moreover, since any fault injected during any time of the computation will cause the change of the temporary result. The adversary cannot retrieve valuable information regarding the secret key bit.

5.2 Security Analysis of Existing Countermeasure Techniques

Although the techniques introduced in previous section are great improvement to the strength of existing algorithms. They still face security challenge stand alone.

5.2.1 High-Order Attack

The high-order attack is first proposed by Muller and Valette in [36]. This attack is specifically derived from the statistical property of the exponent splitting. Although r is a randomly selected number, the pair (r, r') is not uniformly distributed since they satisfy $d = r + r'$.

If the $i - th$ bits of r, r' and d is denoted as r_i , r_i' and d_i, C_i representing the carry bit generated at the $i - th$ iteration. The following equation is satisfied.

$$C_i \oplus r_i \oplus r_i' = d_i$$

Let P_i be the probability that the carry bit is zero, and P_r is set to be the probability the pair (r_i, r_i') be a certain value. In the case $d_i = 0$ and $C_i = 0$, the probability for the pair (r_i, r_i') holds the value $(0,0)$ or $(1,1)$ is the same. When the value is $(0,0)$, it is easy to get $C_{i+1} = 0$. On the contrary, $C_{i+1} = 1$. The same rules apply when $d_i = 1$. Table. 5.1 shows the transition of the probability. The probabilities of iteration i is generated from the previous iteration $i - 1$. It is a Markov chain.

Table 5.1 Probability transition when $d_i = 0$ or 1

$P_r(r_i, r_i')$	$d_i = 0$	$d_i = 1$
$P_r(0,0)$	$0.5 \times P_i$	$0.5 \times (1 - P_i)$
$P_r(0,1)$	$0.5 \times (1 - P_i)$	$0.5 \times P_i$
$P_r(1,0)$	$0.5 \times (1 - P_i)$	$0.5 \times P_i$
$P_r(1,1)$	$0.5 \times P_i$	$0.5 \times (1 - P_i)$
P_{i+1}	$0.5 \times P_i$	$0.5 \times (1 + P_i)$

An example in [36] illustrates the idea of how this attack works. A secret d with length 24 bits is chosen and the probability distribution of the pair (r_i, r_i') is computed in Table.5.2. The table shows whenever the secret bit d has a long run of 0s or 1s, it is very likely the randomly generated pair (r_i, r_i') hold different value. Taking the probability transition illustrated in Table.5.1 into account, the probability of P_i can be calculated. In the case the secret bit is running long 0s (from d_7 to d_{11} in the table), P_i is approaching 0

Table 5.2 An example of bit-level imbalance [36]

	d_0	d_1	d_2	... ...	d_7	d_8	d_9	d_{10}	d_{11}	... ...	d_{18}	d_{19}	d_{20}	d_{21}	d_{22}	d_{23}
(r_i, r_i')	0	1	0	... ...	0	0	0	0	0	... ...	0	1	1	1	1	1
$P_r(0,0)$	50	25	38	... ...	16	8	4	2	1	... ...	8	47	23	11	5	2
$P_r(1,0)$	0	25	12	... ...	34	41	46	48	49	... ...	42	3	27	39	45	48
$P_r(0,1)$	0	25	13	... ...	33	42	46	49	49	... ...	43	4	28	40	46	49
$P_r(1,1)$	50	25	37	... ...	17	9	4	1	1	... ...	7	46	22	10	4	1

indicating that there is no carry bit generated. While in the case of continuous 1s (from d_{19} to d_{23} in the table), P_i gets very close to 1, shows that the carry bit is propagating as the computation runs. If the attacker can launch an attack revealing the probabilities of the pair (r_i, r_i'), he can proceed to find information regarding the secret bit. The secret bit will no longer be secret.

5.2.2 Template Attack

Template attack [37] is based on the fact that power consumptions can be characterized by a multivariate normal distribution during the computations. Normally, it can be executed in two steps. The first step is to setup the templates and the second step is using the template to initialise attacks.

The multivariate normal distribution is defined by a covariance matrix C and a mean vector m. Where C holds the covariance of the targeting point and m is the mean of all the point on the trace. The pair (m, C) is called the template. By sending in different data and key bit, a group of power trace is generated by the adversary. Until then, every data and key pair leads to a template. Then as stated above, in the second step, the attacker

compares the power trace from the target device with all the templates, and determine which one has the highest probability to be the correct template. After the correct template is recovered, the key is then retrieved.

Since in ECC scalar multiplication, the base point P is fixed. This observation offers template attack a great opportunity. In [38], Herbst and Medwed proposed a template attack scheme against masked MPL. In the scenario of scalar multiplication, the attacker set up the templates by running the scalar multiplication several times using different input data. The hamming weight of some intimidate values are also correlated. The power trace is then obtained to match with the template. In order to simplify the matching process, the first two multiplication of the base point are usually taken out to do the match. The template with highest matching probability indicates the first bit. The mask technique only blinds the point, but still by means of template attack, partial bits of secret k is recovered. The attacker can then focus on the masking operation to determine the mask R. Since the hamming weight is known by the matching process. The mask R will not be safe.

5.3 Modified MPL with SM, ES and PR

Since all the countermeasures listed in previous section is vulnerable to different attacks stand alone. He, Huang and Wu burrowed some of the ideas mentioned before and combined them to create a more secure algorithm [11].

5.3.1 Algorithm Explanation

The new algorithm is illustrated as algorithm 5.2. The new algorithm is a combination of three ideas. The sequence masking technique [11], exponent splitting [34] and randomization of the message [35]. The combination of these three techniques greatly improved the security strength of the MPL. Further protections are provided to the

vulnerabilities stated in previous sections.

This algorithm works as follows, in the pre-computation phase, a random number k_0 smaller than secret bit k is generated. The secret bit k is divided into two parts, the first part is the random number k_0, the second part is calculated as $k_1 = k - k_0$. After that a random point R on the same curve with point P is generated. The point R is acting as a mask, it is initialized to all the registers and the value is updated in each iteration. Another random number S with the same bit size of secret key k is generated afterwards. The introduction of random number S is acting as a switch, it determines which computation to be executed. The original scalar multiplication is computed following each bit of k sequentially. With the random number S, it turns into a randomly computed process.

Moving to the main iteration part of the algorithm, registers R_0 and R_1 holds the values related to the secret bit k_0. While R_2 and R_3 store the values computed using k_1. As introduced above, two scalar multiplications take turns to compute according to the current bit of S. When the main iteration part is finished, a final adjustment adding the values in R_0 and R_2 is carried out to get the desired output.

The following observations proved the correctness of the algorithm easily. Let i denotes the iteration number, the mask is updated following a pattern $2^i R$. The first scalar multiplication generates the intermediate value $2^i R + k_0 P$, the second scalar multiplication holds that $2^i(-R) + (k - k_0)P$. Adding them up, we get the value of kP.

Algorithm 5.2. Modified MPL with SM, ES and PR

Input: $P \epsilon E, k = (k_{n-1}, \ldots, k_1, k_0)_2, k_i \epsilon 0,1$

Output: $Q = kP \epsilon E$

Step 1: $Select\ random\ point\ R\ on\ E;$

Step 2: $Select\ random\ number\ k_0 = (k_{n-1}^{(0)} \ldots k_0^{(0)})_2\ \ that\ k_0 \epsilon (1, k)$

$let\ k_1\ (k_{n-1}^{(1)} \ldots k_0^{(1)})_2 = k - \ k_0\ ;$

$k_0\ and\ k_1\ are\ stored\ in\ shift\ registers$

$D_0 = (d_{n-1}^{(0)} \ldots d_0^{(0)})_2\ and\ D_1 = (d_{n-1}^{(1)} \ldots d_0^{(1)})_2$

Step 3: $Genarate\ n\ bit\ sequence\ \mathrm{S} = (s_{n-1}, \ldots, s_1, s_0)_2$

Step 4: Set $R_0 = R, R_1 = P + R, R_2 = -R, R_2 = P + (-R)$

Step 5: $for\ i{=}n{-}1\ down\ to\ 0\ do$

Step 6: $\quad if\ s_i = 0, then$

Step 7: $\quad if\ d_{n-1}^{(0)} = 0, then\ R_1 = R_0 + R_1;\ R_0 = 2R_0;$

Step 8: $\quad else\ \ R_0 = R_0 + R_1;\ R_1 = 2R_1;$

Step 9: $\quad end\ if;\ D_0\ shifts\ left\ by\ 1\ bit.$

Step 10: $else\ if d_{n-1}^{(1)} = 0, then\ R_3 = R_2 + R_3; R_2 = 2R_2;$

Step 11: $\quad else\ R_2 = R_2 + R_3; R_3 = 2R_3;$

Step 12: $\quad end\ if; D_1\ shifts\ left\ by\ 1\ bit.$

Step 13: $end\ if$

Step 14: $end\ for$

Step 15: $for\ i = n - 1\ down\ to\ 0\ do$

Step 16: $\quad if\ s_i = 1, then$

Step 17: $\quad if\ d_{n-1}^{(0)} = 0, then\ R_1 = R_0 + R_1;\ R_0 = 2R_0;$

Step 18: $\quad else\ \ R_0 = R_0 + R_1;\ R_1 = 2R_1;$

Step 19: $\quad end\ if;\ D_0\ shifts\ left\ by\ 1\ bit.$

Step 20: $else\ if d_{n-1}^{(1)} = 0, then\ R_3 = R_2 + R_3; R_2 = 2R_2;$

Step 21: $\quad else\ R_2 = R_2 + R_3; R_3 = 2R_3;$

Step 22: $end\ if; D_1\ shifts\ left\ by\ 1\ bit.$

Step 23: $end\ if$

Step 24: $end\ for$

Step 25: $Return\ R_0 = R_0 + R_2$

5.3.2 Security Analysis

As mentioned in previous sections, this algorithm provides enhanced protection to more attacks compared to existing algorithms.

Since algorithm 5.2 still holds the high regularity MPL offers, SPA is not effective against it. The main computation part always compute a doubling and an addition regardless the input bit. This it keeps all the resistance the regular MPL can provide. C-safe error attack is also eliminated because there is no dummy operation in this modified MPL.

Taking M-safe error into account. In M-safe error attack scheme, the induced memory fault need to be very carefully timed to determine whether it changed the result. Even if the attacker reveals a secret bit by a successfully inducing a safe error, the whole key pattern is not compromised since the key is divided into two parts. Moreover, the computation is executed in a random order. Thus M-safe error cannot threat the modified MPL.

Another attack mentioned above is the comparative power analysis, it also has little effect on the modified MPL with SM, ES and PR. This attack scheme is very similar to the relative doubling attack. Both the two attacks are trying to get two power traces and by comparing them to get valuable information of the secret key. Since the secret key is randomly divided into two parts in every initialization, the two power traces generated will follow different key pattern. It is very hard to make a meaningful comparison between

them. Even if a collision is observed, the mask applied to it in the precomputation will make the power trace totally different. An example listed below in Fig.5.1 will better explain the idea. The table above the power traces show the bits involved in the computation. The top row is the randomly generated sequence s, it is different in every initialization of the process. Below are the corresponding computation decided by sequence s. Secret bit k_0 will be processed when current bit of sequence s holds 0. Otherwise, secret bit k_1 will be computed. Only one of the two computations is carried out at the same time frame.

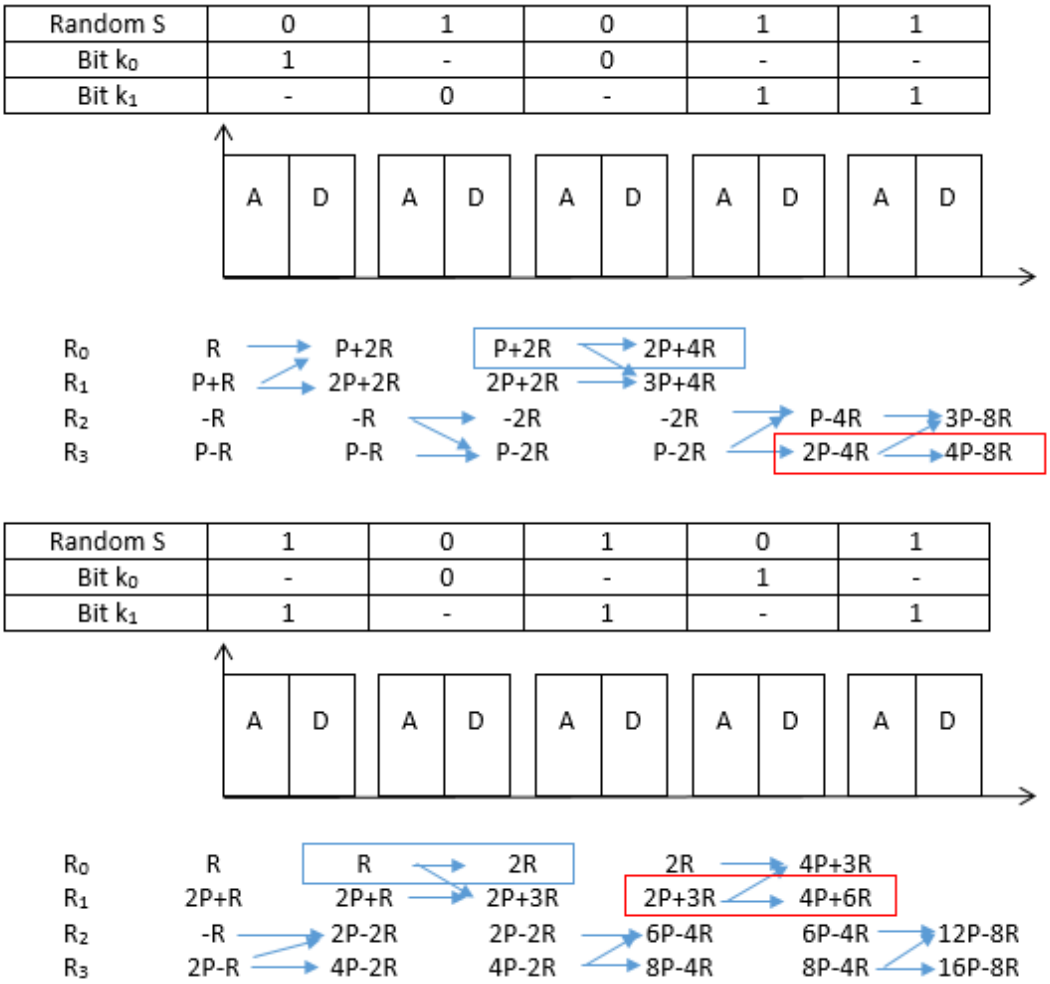

Figure 5.1 Resistance of algorithm 5.2 to relative doubling attack

Taken the adjacent bits in the blue box as the first comparison bit, the computation involved is obviously different. But the adversary cannot easily say that the two bits are different since the two computations are carried out using different key k_0 and k_1, thus comparisons like this will become pointless, it provides no useful information regarding the secret bit. The second sample bits will be taken from the red boxes. Although the unmasked computations are all $2P \rightarrow 4P$, the adversary will still observe different power traces since the mask is different as shown in Fig. 5.1. From above examples, the fact that comparison between adjacent bits cannot offer any useful information is easily obtained. Comparative power analysis shares similar scheme to relative doubling attack. It is also ineffective since Algorithm 5.2 breaks the hidden relationship it relies on.

High-order attack is also worry free. When the attacker is trying to get enough samples to analysis the probability of the imbalance statistic property, the first thing he must do is to get access to the secret key. In the case of Algorithm 5.2, the secret key is randomly divided into two parts. The attacker need to know both parts to start collet samples and do the analysis, so high-order attack need to be combined with other attacks to be effective. As already stated above, SPAs are stopped by the property of highly regular. Relative doubling attack and comparative power analysis are of little use since every power trace generated are randomized by the random sequence s. Fault attack will also be stopped. Even though the attacker uses faults injected to one of the two scalar multiplications to make the two scalar multiplication distinguishable, it is hard for him to find a non-fault reference since all the intermediate values are masked. With above analysis, these attacks mentioned cannot threat Algorithm 5.2. That leaves the last one, template attack.

By building Template attack can comparison the actual computation and the

template and in Section 5.2.2, we introduced how the mask R will be compromised. Assume in Algorithm 5.2, the mask R is known using template attack, then in order to reveal the secret key, the attacker need to get the intermediate values. In other words, he need to compute the updated mask $2^{n-i}R$. For the two scalar multiplications, the iteration number i is different. The iteration number is decided by the random sequence s, which doesn't participate in any computations. It only acts like a switch to decide which scalar multiplication to be computed. Thus even if the attacker retrieved the initial mask R, the updated mask is still a big challenge. Algorithm 5.2 will still be secure.

With all the above analysis, we can see that Algorithm 5.2 is highly secure. It resists SCAs, relative doubling attack, comparative power analysis, high-order attack and template attack. It greatly enhanced the security strength of MPL from algorithm level. Thus it is chosen as the implementation algorithm.

CHAPTER VI

PROPOSED HARDWARE IMPLEMENTATION

In this chapter, based on previous Algorithm 5.2, an efficient hardware architecture is proposed and its FPGA implementation is presented. Very high speed integrated circuit (VHSIC) Hardware Description Language is chosen as the target implementation language. A modern Xilinx Virtex 7 (XC7VX690TFFG1926-3) field-programmable gate array (FPGA) device is used in the implementation. The ECC parameters are NIST-recommended elliptic curve for $GF(2^{233})$ in [39], as shown in Table. 6.1, where $f(x)$ is the irreducible polynomial, n is the order, G_x and G_y are base point coordinates.

NIST-recommended elliptic curve for GF(2^{233})
Elliptic Curve $E: y^2 + xy = x^3 + ax^2 + 1, a = 0$
$f(x) = x^{233} + x^{74} + 1$
$n = 8000000000000000000000000000069d5bb915bcd46efb1ad5f173abdf$
$G_x = 17232ba853a7e731af129f22ff4149563a419c26bf50a4c9d6eefad6126$
$G_y = 1db537dece819b7f70f555a67c427a8cd9bf18aeb9b56e0c11056fae6a3$

Table 6.1 NIST-recommended parameters

6.1 Implementation Hierarchy of the ECC operations

The building blocks of computation involved in ECC is illustrated in Fig. 6.1. Finite field arithmetic such as field addition, subtraction, multiplication, inversion and squaring are the fundamental computations. Both elliptic curve point addition and doubling are based on the finite field computations. As the figure shown, the upper layer computations

are constructed by the lower layers. Scalar multiplication is realized by different algorithms based on point addition and doubling. Elliptic curve cryptographic schemes such like ECDSA are on the top.

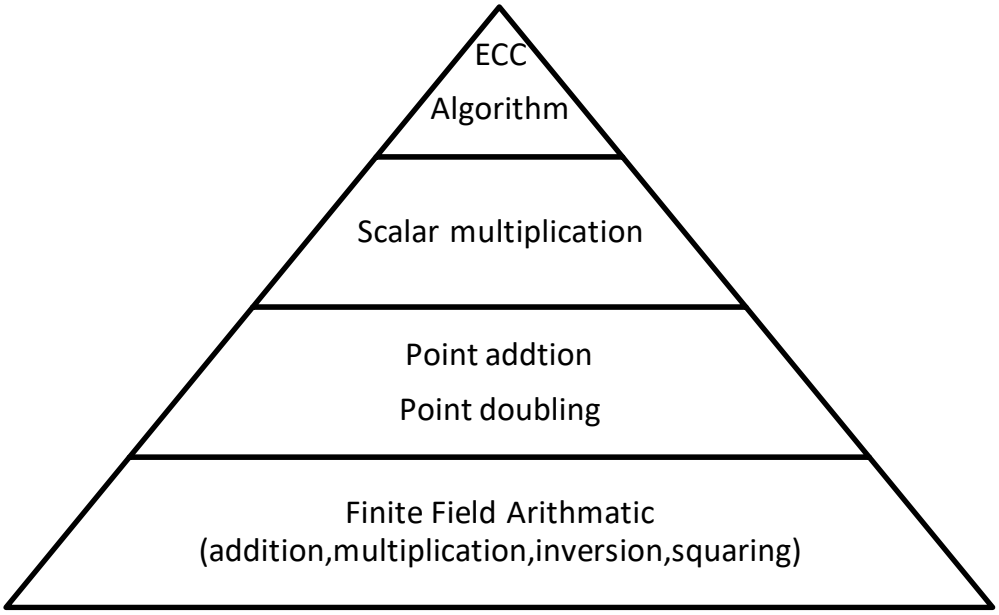

Figure 6.1 Hierarchical architecture for computation involved in ECC

The main components of this ECC design are: field multiplication, field squaring, field inversion, group operations and random number generation. Recall that we introduced the polynomial basis in Section. 2.1. All implementation module in this chapter are using the polynomial basis representation.

6.2 Random Number Generation

In the pre-computation part, there three random binary sequence need to be generated. This process is implemented with a linear feedback shift register (LFSR). A LFSR is s sequential shift register with combinational logic that causes it to pseudo-

randomly cycle through a sequence of binary values. It has well-known applications in generating pseudo-random binary sequence. A pseudo-random binary sequence is considered pseudo because it will start to repeat the pattern after a certain number of states. In order to make the generation more close to a real random number, the LFSR need to reach its maximum length. In other words, an n bit LFSR need to generate all 2^n -1 states before it starts to repeat itself. By carefully chosen the positions of the bits feeding back to the next state, a maximum length LFSR can be achieved.

For the case of the 233 bit random sequence, the tap value is 233 and 159 [40]. There are two structures of the LFSR. One is one-to-many structure (also known as Galois LFSR). The other is many-to-one structure (also known as Fibonacci LFSR). As Fig.6.2 illustrated, a 233-bit Galois LFSR is built. This structure is chosen rather the many-to-one structure, is because that Galois LFSR generates all the feedback bits parallel. In this way, the LFSR runs more efficiently.

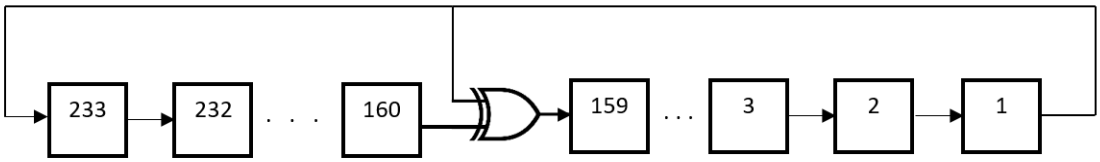

Figure 6.2 233-bit LFSR

The LFSR will generate three random sequence. The first random number will continue generating until it is smaller than order of base point P, then it is assigned to k_0 according to Algorithm 5.2. The second random sequence will generate and then assigned to s, acting as the switch to determine which scalar multiplication is going to perform. The third random sequence will be r, it will be used in the precomputation to compute rP. Since

r is random, rP will be random point. The computation will need the modules introduced in later sections.

6.3 Addition in $GF(2^m)$

Field addition is very easy to implement in VHDL. As stated in Section. 2.1, field addition is simply a bit-wise exclusive-or in either hardware or software. Subtraction in $GF(2^m)$ is the same as addition in $GF(2^m)$, since the additive inverse of an element is it-self. All finite field addition is realized using simple x-or gate.

6.4 Multiplication in $GF(2^m)$

The classical way to implement multiplication is the two-step computation. To perform a finite field multiplication, the first step is to do a multiplication and the second step is reduction. Pamula introduced another basic architecture in [41]. It is called the interleaved multiplication. In this method, the multiplication and reduction are interleaved. It is based on the following observation. Given two polynomials,

$$a(x) = a_{m-1}x^{m-1} + \cdots a_1 x + a_0$$

$$b(x) = b_{m-1}x^{m-1} + \cdots b_1 x + b_0$$

and define the irreducible polynomial

$$f(x) = x^m + f_{m-1}x^{m-1} + \cdots f_1 x + f_0$$

the product will be given by

$$c(x) = a(x)b(x)\ mod\ f(x) = a(x)\sum_{0}^{m-1} b_i x^i\ mod f(x)$$

$$c(x) = (b_0 a(x)x + b_1 a(x)x^2 + \cdots + b_{m-1}a(x)x^{m-1}) mod\, f(x)$$

$a(x)x$ can be substituted by the following equation

$$a(x)x = d = a_{m-1}x^m + \cdots a_1 x^2 + a_0 x$$

$$d = d_{m-1}x^{m-1} + \cdots d_1 x + d_0 \text{ where } \begin{cases} d_0 = a_{m-1}f_0 \\ d_i = a_{i-1} + a_{m-1}f_i \end{cases}$$

By applying these equations, the following structure is implemented. In the right part of Fig.6.3, a partial result interleaved with reduction is calculated. The output is used as the input in next iteration and sent to the left to do the accumulation. The shift register holds the value of b and shifts right in every iteration. The left part accumulate the partial product in every iteration. And after m iterations, the result is computed.

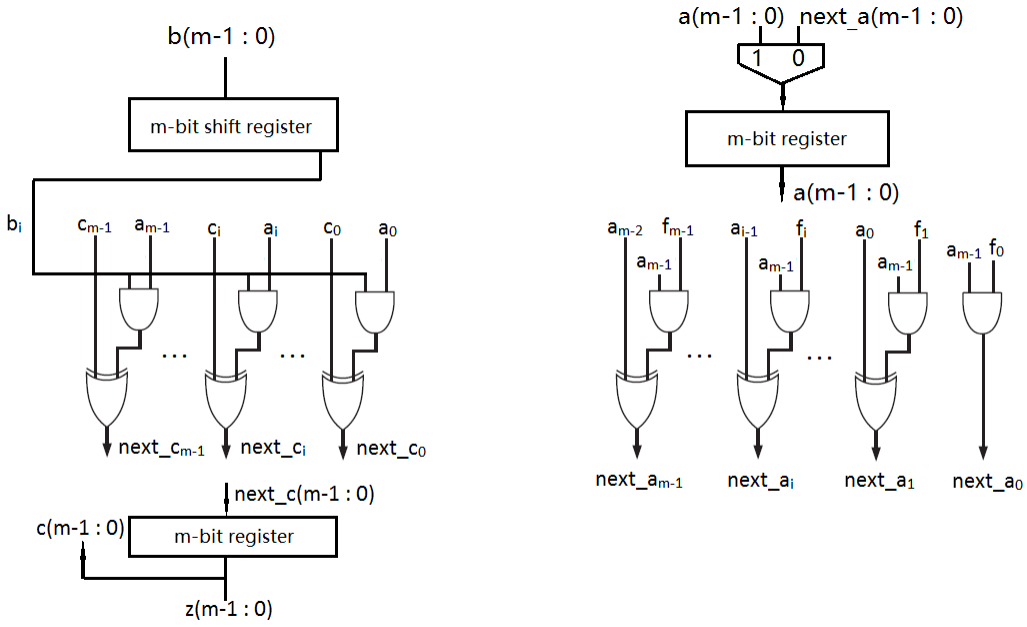

Figure 6.3 Multiplication unit

6.5 Squaring in $GF(2^m)$

Polynomial basis squaring is relatively simple compared with multiplication. It is very similar to the classical two-step multiplication. The squaring operation can be done in two steps. The first step is to insert a 0 bit between consecutive bits of the binary

representation. The second step is also the polynomial reduction. In Fig.6.4, a pattern for squaring a polynomial $a(x)$ is shown.

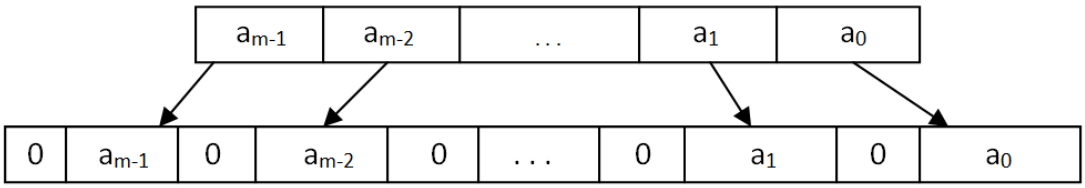

Figure 6.4 Squaring a polynomial

The reduction of it can however be achieved using a method called the reduction matrix [41]. The reduction matrix is constructed using the irreducible polynomial and can be computed when the operation starts. Since the polynomial representation of the squaring result before reduction is easy to get. Applying the reduction matrix method can greatly speed up the reduction process.

6.6 Inversion in $GF(2^m)$

Inversion is the most time consuming operation among all the implementation module. There are serval existing method to do the field inversion over $GF(2^m)$, the Fermat's method and the extended Euclidean algorithm. While the Fermat's method takes lots of time to calculate the value, here we choose the extended Euclidean algorithm [42]. Euclid's algorithm is for calculating the greatest common divisor of two polynomials. The algorithm is extended to find two polynomials satisfying that

$$\gcd\big(a(x),b(x)\big) = u(x) \times a(x) + w(x) \times b(x)$$

If $a(x)$ is an element of the field defined by irreducible polynomial $f(x)$, we had the relation that gcd $(a(x),f(x)) = 1$. By replacing $b(x)$ with $f(x)$, the above equation is deduced to $1 = u(x) \times a(x)\ mod\ f(x)$. The inverse of $a(x)$ can then be calculated by

$$a(x)^{-1} = u(x)\ mod\ f(x)$$

Algorithm.6.1 illustrates how the inverse is calculated, the implementation is straight forward while the addition is done using x-or, division is done by right shift and multiplication is done by left shift. The result is outputted after 2m iterations.

Algorithm 6.1. Extended Euclidean Method [42]

Input: $a(x), f(x)$

Output: $u(x) = a(x)^{-1}$

Step 1: $let\ s(x) = f(x), v(x) = 0, r(x) = a(x), u(x) = 1, d = 0;$

Step 2: $for\ i{=}0\ to\ 2\mathrm{m}\ do$

Step 3: $if\ r_m = 0, then\ r(x) = xr(x), u(x) = xu(x), d = d + 1$

Step 4: $else\ if\ s_m = 1,$

Step 5: $then\ s(x) = s(x) - v(x), v(x) = v(x) - u(x)$

Step 6: $end\ if$

Step 7: $s(x) = xs(x)$

Step 8: $if\ d = 0,$

Step 9: $then\ r(x) = s(x), s(x) = r(x)$

Step 10: $u(x) = xv(x), v(x) = u(x)$

Step 11: $d = 1$

Step 12: $else\ u(x) = \frac{u(x)}{x}, d = d - 1$

Step 13: $end\ if$

Step 14: $end\ if$

Step 15: $end\ for$

6.7 Elliptic Curve Group Operations in GF(2^m)

The group operations in ECC are the point addition and point doubling operations. As illustrated in Section 2.2, the group operations are implemented using the previous modules. Fig. 6.5 shows the block diagram of the point addition operation.

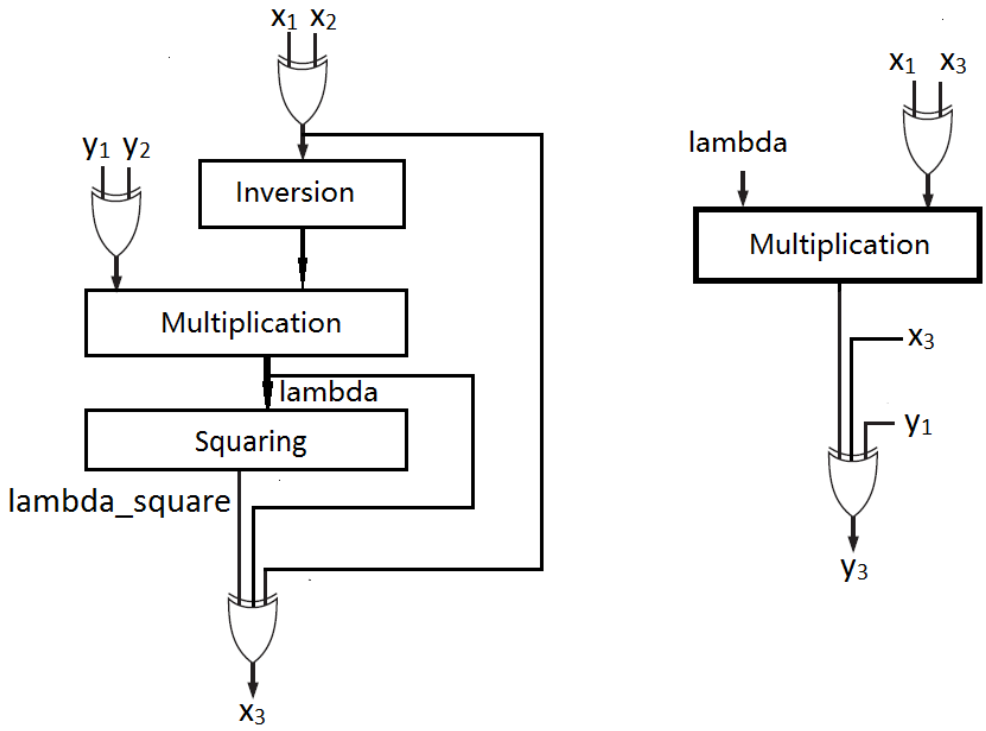

Figure 6.5 Hardware architecture of point addition

It can be seen that as the equation in Section 2.2 shows, the point addition operation module requires two multiplications, one squaring and one inversion.

Similarly, Fig 6.6 shows the architecture of point doubling module. It requires two multiplications, two squaring and one inversion. The main computation part of Algorithm 5.2 is building using these two modules, in the next section. We will discuss how the modified MPL with SM, ES and PR is realized.

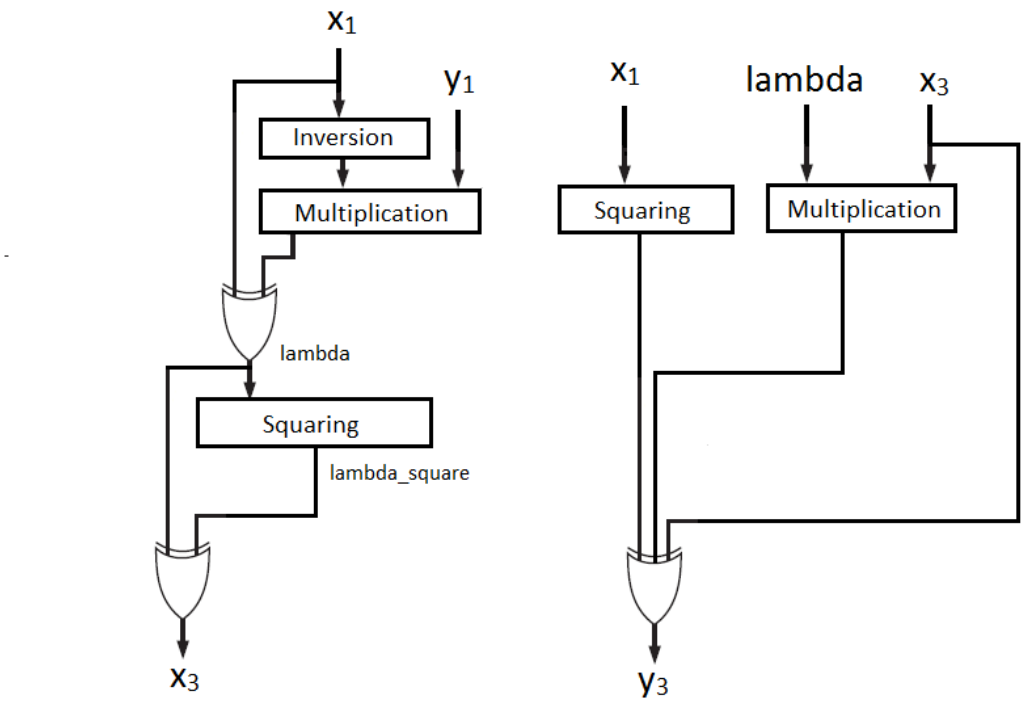

Figure 6.6 Hardware architecture of point doubling

6.8 Scalar Multiplication

As the main computation of the ECC, Algorithm 5.2 provides a whole new MPL with strong resistance to SCAs. During the pre-computation process, the first scalar random sequence k_0 is generated. The second scalar k_1 is computed by $k - k_0$ .Another random sequence s is produced and finally the random point $R = rP$ is acting as the mask.

There four registers R_0, R_1, R_2 and R_3 holding 4 different intermediate values. They are initialized by the x, y coordinates of the base point P and mask R. R_0 and R_1 are given the value R and $P + R$ at the beginning. R_2 and R_3 are initialized with $-R$ and $P - R$. The anti-mask $-R$ shares the same x coordinate with R. While the y coordinate is calculated simply x-or the coordinates of mask R. Three shift registers to store the value of s, k_0, and k_1. The control unit is the core unit to realize the algorithm. The input and the

output of the point addition and point doubling module is controlled by this unit. The control unit has 36 different states to decide which value from which register to be given as the input of point operation module, also at the same time, the specific register to store the output is decided. The states are controlled by the three random values in the shift register. Additionally, a counter aiming for m cycles is built in as to tell the whole process when to stop the computation. Fig. 6.7 illustrates the blocking diagram of the top main computation module.

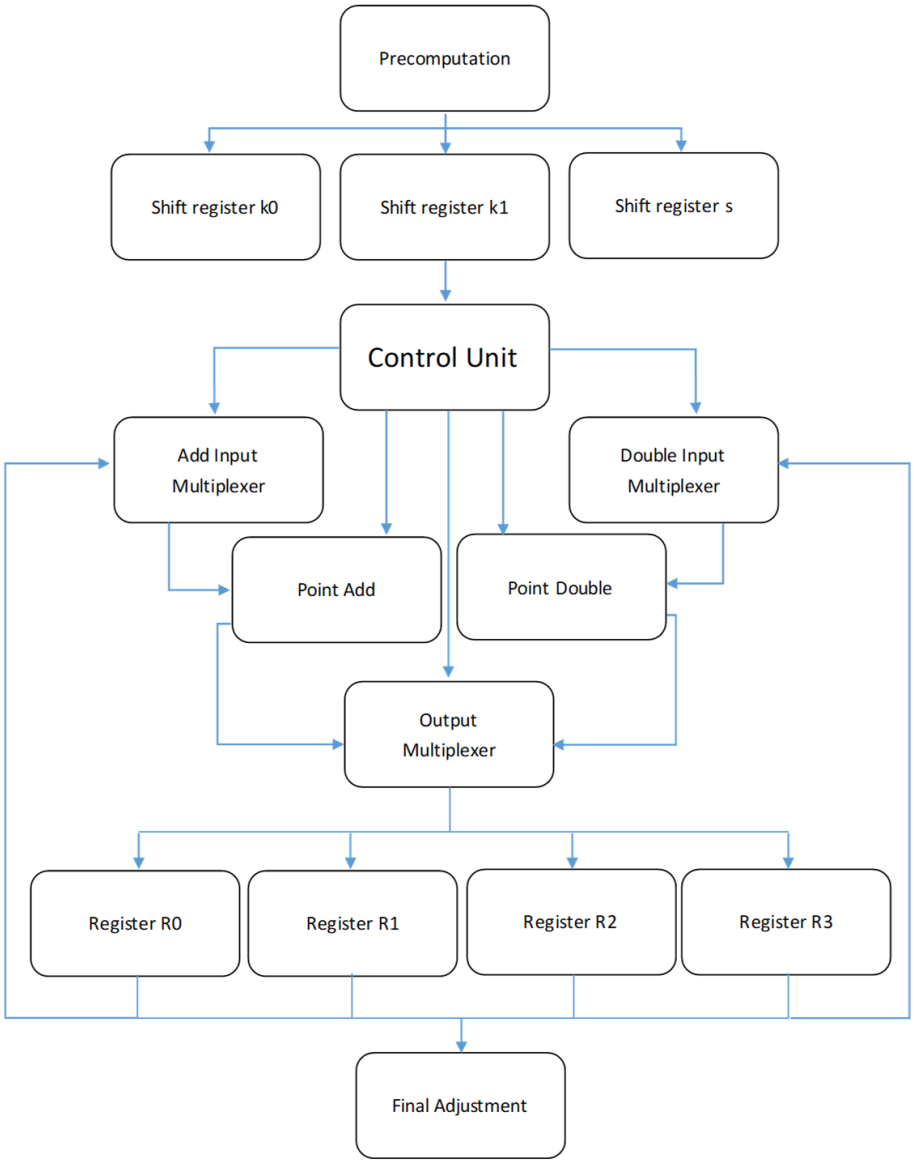

Figure. 6.7 Architecture of proposed implementation

6.9 Synthesis Results

The VHDL code is synthesized for Xilinx XC7VX690 using Vivado 2017. The hardware resource usage is summarized in Table. 6.2. The computation time at 100 MHz is 4.43 ms. We can see Algorithm 5.2 doubles the computation time needed since it consist of two scalar multiplication. Fig. 6.8 shows the result of the output waveform.

Algorithm	Number of FFs	Number of LUTs	Number of IOs	Clock Cycles
Regular MPL	8317	8753	708	220,020
Algorithm 5.2	11247	11405	708	442,493

Table 6.2 Hardware usage of different algorithms implemented

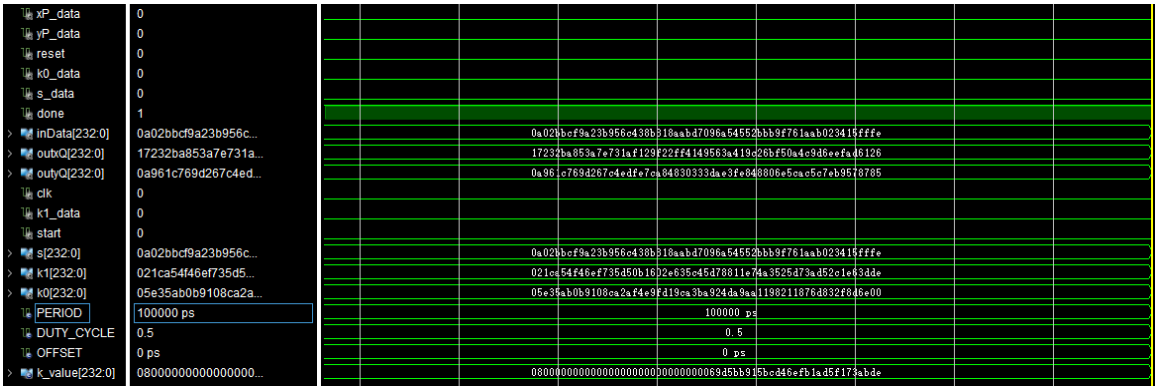

Figure 6.8 Output waveform

The hardware implementation results and performance comparisons with some existing implementations are listed in Table. 6.3. It is to be noted that since the result provided in literature are implemented on different FPGA technologies from out design. Thus, a straight forward comparison is hard to make.

Work	Algorithm	Device	Field	FFs	LUTs	f(MHz)	Time	Cycles
[32]	MPL	Spartan-3	$GF(2^{163})$	3265		130	1 ms	n.a.
[44]	Binary	XC4VLX80	$GF(2^{163})$	24,263		143	n.a.	n.a.
[45]	Binary	XC2V6000	$GF(2^{233})$	16970	19,440	100	n.a	n.a.
[46]	MPL	XCV2000E-7	$GF(2^{233})$	10632	35,800	67.9	n.a	n.a.
[47]	Binary	XCV2000E	$GF(2^{233})$	15,478		37	13.2 ms	n.a.
[29]	NAF	XC3S1000	$GF(2^{233})$	n.a.	n.a.	80	2.28 ms	183000
[24]	Binary	XC7K325T	$GF(2^{233})$	9407	9151	255.66	2.66 ms	679776
Proposed	MPL	XC7VX690	$GF(2^{233})$	8317	8753	100	2.2 ms	220020
Proposed	Modified MPL	XC7VX690	$GF(2^{233})$	11252	10405	246.1	1.8 ms	442493

Table 6.3 FPGA implementation complexity comparison between proposed design and related works

From the table, it is clear that there are very few implementation done with countermeasures, the main concern for most of the designs listed are speed. [4, 26, 41 and 43] adapted the projective coordinates to reduce the amount of inversion operation and improve the calculation speed. Scalar multiplications in [32] uses MPL without any

countermeasure. Implementations in [21, 41 and 42] uses binary method for scalar multiplication. All these mentioned references are applying the algorithms without any further protections.

Algorithm/Implementation	Doubling [8]	Relative doubling [30]	Comparative power analysis [31]	M-safe error [19]	C-safe error [9]	High-order [36]	Template attack [37]
Binary [21,41,42,44]	n.a	n.a	n.a	n.a	n.a	n.a	n.a
NAF [29]	×	n.a	×	n.a	n.a	n.a	n.a
MPL [29,43]	√	×	×	×	√	n.a	n.a
Masked MPL [11]	√	n.a	√	√	n.a	n.a	×
Exponent Splitting [11]	√	n.a	√	√	n.a	×	n.a
Proposed	√	√	√	√	√	√	√

Table 6.4 Comparison of SCA countermeasure property between proposed work and existing related works

Our proposed implementation on scalar multiplication is not as efficient as some of the listed design such like [4 and 26]. But our implementation mainly focus to provide a protected power trace to eliminate potential SCA threat. And the clock cycle is comparable to some listed design such like [24]. Table. 6.4 shows that our implementation can resist

most existing side channel attacks comparing with related implementations. Overall, our implementation can provide better protection against SCAs.

CHAPTER VII

DISCUSSION AND POSSIBLE FUTURE WORKS

7.1 Discussion

In this thesis, an efficient architecture for the scalar multiplication algorithm [11] is proposed. A FPGA implementation of the algorithm [11] is presented. It is the first time that this algorithm is implemented in hardware.

This implementation resistant to most existing side channel attacks such as doubling attack [8], relative doubling attack [30], comparative power analysis [31], m-safe error attack [19], c-safe error attack [9], high-order [36] and template attack [37]. As shown in Table 6.4, compared to the existing related works, the proposed implementation offers the best countermeasures to SCAs.

7.2 Possible Future Work

As a pseudo-random number generator, LFSR is simple and fast but its output does not have the property of very good randomness. It follows a pattern that can repeat after a certain number of states. Those sequences of numbers are random-like in some aspects. If the attacker knows the seed and also the tap values, the randomness of the generated sequence maybe compromised. A better random generator such like mentioned in [43] can further protect the implementation.

In addition, since our design is implemented using affine coordinates, projective coordinates [29] can be adopted in the design. The advantage of using projective coordinates is that the amount of finite field inversion operation can be greatly reduced with proper pre-computation. Finite field inversion operation is considered as the most time

consuming module in ECC scalar multiplication. So the computation time may be shortened if projective coordinate systems is adopted.

Moreover, this design features a regular bit serial interleaved multiplier. Faster method for implementation such like digit level multiplier in [44] can be utilized to further speed up the elliptic curve scalar multiplication.

REFERENCES

[1] V.S. Miller, "Use of Elliptic Curves in Cryptography," *Advances in Cryptology Proc. (CRYPTO'85), Springer-Verlag, LNCS 218*, pp. 417-426, 1985.

[2] N. Koblitz, "Elliptic Curve Cryptosystems," *Mathematics of Computation*, vol. 148, pp. 203-209, 1987.

[3] W. Diffie and M.E. Hellman, "New directions in cryptography," *IEEE Transaction of Information Theory*, vol. 22, pp. 444-454, 1976.

[4] G. Sutter, J. Deschamps, and J. Imana, "Efficient elliptic curve point multiplication using digit-serial binary field operations," *IEEE Transactions on Industrial Electronics*, vol. 60, pp. 217-225, Jan. 2013.

[5] P. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," *Advances in Cryptology Proc. (CRYPTO'96), Springer-Verlag, LNCS 1109*, pp. 104-113, 1996.

[6] P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," *Advances in Cryptology Proc. (CRYPTO'99), Springer-Verlag, LNCS 1666*, pp. 388-397, 1999.

[7] J.S. Coron, "Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems," *Proc. Int'l Cryptographic Hardware and Embedded Systems (CHES '99)*, pp. 192-302, Aug. 1999.

[8] A.P Fouque and F. Valette, "The Doubling Attack: Why Upwards is Better than Downwards," *Proc. Int'l Workshop Cryptographic Hardware and Embedded Systems (CHES'03)*, pp.269-280, Sept. 2003.

[9] M. Joye and S.M. Yen, "The Montgomery Powering Ladder," *Cryptographic Hardware and Embedded Systems (CHES'02)*, pp. 291-302, 2002.

[10] P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," *Advances in Cryptology Proc. (CRYPTO '99), Springer-Verlag, LNCS 1666*, pp. 388-397, 1999.

[11] Y. He, "Highly Secure Cryptographic Computations against Side-Channel Attacks," Master thesis, University of Windsor, 2012.

[12] M. Abdalla, M. Bellare, and P. Rogaway, "DHAES: An encryption scheme based on the Diffie-Hellman problem," *IEEE P1363a*, 1998.

[13] V. S. Miller, "Use of elliptic curves in cryptography," *Abstracts for Crypto '85*.

[14] D. Johnson and A. Menezes, "The elliptic curve digital signature algorithm (ECDSA)," *Technical report CORR 99-34, Dept. of C&O*, University of Waterloo, 1999.

[15] Y. Li, M. Chen, and J. Wang, "Introduction to Side-Channel Attacks and Fault Attacks," *Proc. Asia-Pacific Symp. Electromagn. Compat. (APEMC)*, vol. 1, pp. 573-575, May. 2016.

[16] D. Boneh, R.A. DeMillo and R.J. Lipton, "On the Importance of Checking Cryptographic Protocols for Faults," *Proc. of EUROCRYPT' 97, LNCS 1233*, pp. 37-51, 1997.

[17] Y. Zhou, and D. Feng, "Side-Channel Attacks: Ten Years after Its Publication and the Impacts on Cryptographic Module Security Testing" *IACR Cryptology ePrint Archive*, 2005.

[18] I. Biehl, B. Meyer and V. Műller. "Differential fault attacks on elliptic curve cryptosystems," *Advances in Cryptology Proc. (CRYPTO 2000), LNCS 1880*, pp.131–146, 2000.

[19] S.M. Yen and Marc Joye, "Checking before Output May not be Enough Against Fault-based Cryptanalysis," *IEEE Transactions on Computers*, vol.49, pp.967-970, 2000.

[20] J. Quisquater and D. Samyde, "Electro Magnetic Analysis (EMA): Measures and Countermeasures for Smart Card," *E-smart 2001*, pp.200-210, 2001.

[21] K. Wu, H. Li, T. Chen, and F. Yu, "Electromagnetic analysis on elliptic curve cryptosystems: Measures and counter-measures for smart cards," *Proc. 3rd Int. Symp. IITA*, vol. 1, pp. 40–43, 2009.

[22] H. Mahanta, A. Azad, and A. Khan, "Power Analysis Attack: A Vulnerability to Smart Card Security," *Proc. Signal Processing and Communication Engineering Systems (SPACES)*, 2015.

[23] D.M. Gordon. "A Survey of Fast Exponentiation Methods," *Journal of Algorithms*, vol. 27, pp. 129-146, 1998.

[24] M.S. Hossain, E. Saeedi, and Y. Kong, "High-Speed, Area-Efficient, FPGA-Based Elliptic Curve Cryptographic Processor over NIST Binary Fields," *Proc.2015 IEEE International Conference on Data Science and Data Intensive Systems*, Dec, 2015.

[25] P.A. Fouque and F. Valette, "The doubling attack–why upwards is better than downwards," *Proc. Int'l Workshop Cryptographic Hardware and Embedded Systems (CHES'03). Springer*, pp. 269–280, 2003

[26] S.M. Yen et al., "A Countermeasure against One Physical Cryptanalysis May Benefit another Attack," *Information Security and Cryptology (ICISC'01), Springer-Verlag, LNCS 2288*, pp. 417-427, 2002.

[27] F. Morain and J. Olivos, "Speeding up the computation on an elliptic curve using addition-subtraction chains," *Inform Theory*, vol. 24, pp.531–543, 1990.

[28] D. Hankerson, A. Menezes and S. Vanstone, "Guide to Elliptic Curve Cryptography," *Springer-Verlag,* 2004.

[29] Y. B. Wang, X. J. Dong and Z.G. Tian, "FPGA based design of elliptic curve cryptography coprocessor," *Third International Conference on Natural Computation, ICNC 2007*, pp. 185-189, Aug, 2007.

[30] S.M Yen et al., "Relative Doubling Attack Against Montgomery Ladder," *ICISC 2005 Springer-Verlag, LNCS 3935*, pp. 117-128, 2006.

[31] N. Homma, A. Miyamoto, T. Aoki, A. Satoh, and A. Samir, "Comparative power analysis of modular exponentiation algorithms," *Computers, IEEE Transactions on*, vol. 59, no. 6, pp. 795-807, 2010.

[32] J. P. Deschamps and G.Sutter, "Elliptic-curve Point-Multiplication Over GF(2^{163})," *Proc. IEEE Conf. on PL*, pp. 25-30, 2008.

[33] A.J. Menezes, "Elliptic Curve Public Key Cryptosystems", *Kluwer Academic Publishers*, 1993.

[34] C. Claver and M. Joye, "Universal Exponentiation Algorithm: a First Step towards Provable SPA-Resistance," *Proc. Int'l Workshop Cryptographic Hardware and Embedded Systems (CHES '01),* pp. 300-308, 2001.

[35] G. Fumaroli and D. Vigilant, "Blinded Fault Resistant Exponentiation," *Fault Diagnosis and Tolerance in Cryptography, Lecture Notes in Computer Science*, vol. 4236, pp. 62-70, 2006.

[36] F. Muller and F. Valette, "High Order Attack against Exponent Splitting Protection," *Proc. Int'l Conference on Practice and Theory in Public-Key Cryptography (PKC '06), Springer-Verlag, LNCS 3958*, pp. 315-329, 2006.

[37] M. Medwed and E. Oswald, "Template Attacks on ECDSA," in *Information Security Applications, WISA*, vol. 5379, pp. 14-27, 2008.

[38] C. Herbst and M. Medwed, "Using Templates to Attack Masked Montgomery Ladder Implementations of Modular Exponentiation," *Proc. Int'l Workshop on Information Security Application* (*ISA '08*), *Springer-Verlag, LNCS 5379*, pp. 1-13, 2009.

[39] P. FIPS, "186-3 Digital Signature Standard (DSS),"*National Institute of Standards and Technology (NIST)*, 2009.

[40] R. Ward and T. Molteno. "Table of Linear Feedback Shift Registers," http://www.eej.ulst.ac.uk/~ian/modules/ EEE515/files/old_files/lfsr/lfsr_table.pdf, 2007.

[41] D. Pamula, "Arithmetic operators on GF(2^m) for cryptographic applications: performance - power consumption security tradeoffs," PhD thesis, Silesian University of Technology; University of Rennes 1, Poland, France, 2012.

[42] J.H. Guo and C.L. Wang, "Systolic array implementation of euclid's algorithm for inversion and division in GF(2^m)," *IEEE Trans. Comput.*,vol. 47, no. 10, pp. 1161-1167, Oct, 1998.

[43] K.Wold and C.H. Tan, "Analysis and Enhancement of Random Number Generator in FPGA Based on Oscillator Rings," *International Journal of Reconfigurable Computing*, June, 2009.

[44] H. M. Choi, C. P. Hong and C. H. Kim, "High performance elliptic curve cryptographic processor over GF(2^{163})," *Proc.4th IEEE Int. Symp. Electron. Design, Test Appl. (DELTA)*, pp. 290-295, Jan, 2008.

[45] C. Grabbe, M. Bednara, J. von zur Gathen, J. Shokrollahi and J. Teich, "A high performance vliw processor for finite field arithmetic", *Reconfigurable Architectures Workshop (RAW)*, 2003.

[46] C. Shu, K. Gaj and T. El-Ghazawi, "Low latency elliptic curve cryptography accelerators for NIST curves on binary fields," *IEEE Field-Programmable Technology (FPT)*, pp. 309–310, 2005.

[47] T. Kerins, E. M. Popovici and W. P. Marnane. "An FPGA Implementation of a Flexible, Secure Elliptic Curve Cryptography Processor," *International Workshop on Applied Reconfigurable Computing-ARC 2005, IADIS press*, pp.22-30, 2005.

APPENDICES

SELECTED VHDL PROGRAMMING CODES

```
----------------------------------------------------------------------------
-- Top level: Introduces the point coordinates and k through the same port
--Author: Che Chen
----------------------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

package my_package is
constant m: natural := 233;
constant logm: natural := 8;
constant zero: std_logic_vector(m-1 downto 0) := (others => '0');
end my_package;

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
use work.my_package.all;

entity top_MaskedMont is
port (
 inData: in std_logic_vector(m-1 downto 0);
--  xP_data, yP_data, k_data: in std_logic;
 clk, reset, start: in std_logic;
 outxQ, outyQ: inout std_logic_vector(m-1 downto 0);
--  xQ_or_yQ: in std_logic;
 done: out std_logic
 );
end top_MaskedMont;

architecture circuit of top_MaskedMont is

 component MaskedMont is
 port (
  xP, yP, xR, yR, y_minus_R, x_pplusr, y_pplusr, x_pminusr, y_pminusr: in std_logic_vector(m-1
downto 0);
          k0, k1, s: in std_logic_vector (m-1 downto 0);
  clk, reset, start: in std_logic;
  xQ, yQ: inout std_logic_vector(m-1 downto 0);
  done: out std_logic
  );
 end component MaskedMont;
```

```
 component rand_gen is
 Port (  clk : in  STD_LOGIC;
                              reset : in  STD_LOGIC;
       start : in  STD_LOGIC;
                              K : in  std_logic_vector (m-1 downto 0);
       k1,s,k0,r : out  std_logic_vector (m-1 downto 0);
       done : out  STD_LOGIC);
 end component rand_gen;

 component EC_montgomery_multiplication  is
 port (
  xP, yP, k: in std_logic_vector(m-1 downto 0);
  clk, reset, start: in std_logic;
  xQ, yQ: inout std_logic_vector(m-1 downto 0);
  Q_infinity: inout std_logic;
  done: out std_logic
);
end component EC_montgomery_multiplication;

component Point_add_v2 is
port(
 x1,y1,x2,y2: in std_logic_vector(m-1 downto 0);
 clk, reset, start: in std_logic;
 x3: inout std_logic_vector(m-1 downto 0);
 y3: out std_logic_vector(m-1 downto 0);
 done: out std_logic
 );
end component Point_add_v2;

 signal xP, yP, k0, k1, s, k, xQ, yQ, r, xR, yR, y_minus_R, x_pplusr, y_pplusr, x_pminusr, y_pminusr:
std_logic_vector (m-1 downto 0);

 signal start_ran, start_main, ran_done, main_done, xP_data, yP_data, k_data, start_r_point,
r_point_done,
 start_add1, add_done1, start_add2, add_done2: std_logic;

 subtype states is natural range 0 to 14;

 signal current_state: states;

begin
          ran: rand_gen port map(
             clk, reset, start_ran,k,
             k1, s, k0, r, ran_done );

randomR: EC_montgomery_multiplication  port map(
                     xP, yP, r,
                                                  clk, reset, start_r_point,
```

```
                                                    xR, yR,
                                                    done => r_point_done);

                                                    y_minus_R <= xR xor yR;

P_plus_R: Point_add_v2  port map(xP, yP, xR, yR, clk, reset, start_add1, x_pplusr, y_pplusr,
add_done1);
P_minus_R: Point_add_v2 port map(xP, yP, xR, y_minus_R, clk, reset, start_add2, x_pminusr,
y_pminusr, add_done2);


MaskedMont port map(
            xP, yP, xR, yR, y_minus_R, x_pplusr, y_pplusr, x_pminusr, y_pminusr,k0, k1, s,
            clk, reset, start_main,
            xQ, yQ, main_done );

registers: process(clk)
 begin
 if clk' event and clk = '1' then
   if xP_data = '1' then xP <= inData; end if;
   if yP_data = '1' then yP <= inData; end if;
   if k_data = '1'  then k <= inData; end if;
 end if;
 end process;

control_unit: process(clk, reset, current_state)
begin

end process;

          outxQ <= xQ; outyQ <= yQ;

end circuit;

----------------------------------------------------------------------------
-- Random number generator
----------------------------------------------------------------------------
entity rand_gen is
   Port ( Clk : in  STD_LOGIC;
                              reset : in  STD_LOGIC;
       start : in  STD_LOGIC;
                              K : in  std_logic_vector (232 downto 0);
       k1,s,k0,r : out  std_logic_vector (232 downto 0);
       done : out  STD_LOGIC);
end rand_gen;

architecture Behavioral of rand_gen is

type state_type is (idle,s0,s1,s2);
```

```
signal state : state_type := idle;
signal rand_out : unsigned(232 downto 0);

component lfsr is
port (clk,reset : in std_logic;
    rand_out : out unsigned(232 downto 0)
  );
end component;

begin

lfsr_inst : lfsr port map(Clk,'0',rand_out); --keep generating random numbers.

process(Clk,reset)
begin
        if(reset = '1') then
                state <= idle;
                done <= '0';
        elsif(rising_edge(Clk)) then
                case state is
                        when idle =>
                                if(start = '1') then --start the fsm.
                                        state <= s0;
                                end if;
                                done <= '0';
                        when s0 =>
                                if(rand_out < unsigned(K)) then --check if k0 is less than K
                                        k0 <= std_logic_vector(rand_out);
                                        k1 <= std_logic_vector(unsigned(K)-rand_out);  --if yes, so
the subtraction and assign it to k1.
                                        state <= s1;
                                else
                                        state <= s0;
                                end if;
                        when s1 =>
                                if(rand_out < unsigned(K)) then --check if k0 is less than K
                                        r <= std_logic_vector(rand_out);
                                        state <= s2;
                                else
                                        state <= s1;
                                end if;
                        when s2 =>
                                s <= std_logic_vector(rand_out); --the next random number is
assigned to s.
                                state <= idle;
                                done <= '1'; --done signal is asserted and goes back to idle state
waiting for next start signal.
                end case;
        end if;
```

```
end process;

end Behavioral;

entity MaskedMont is
port (
xP, yP, xR, yR, y_minus_R, x_pplusr, y_pplusr, x_pminusr, y_pminusr: in std_logic_vector(m-1
downto 0);
k0, k1, s: in std_logic_vector (m-1 downto 0);
clk, reset, start: in std_logic;
xQ, yQ: out std_logic_vector(m-1 downto 0);
--Q_infinity: inout std_logic;
done: out std_logic
);
end MaskedMont;

architecture arch of MaskedMont is

constant zero: std_logic_vector(m-1 downto 0) := (others => '0');

component Point_add_v2 is
port(
 x1,y1,x2,y2: in std_logic_vector(m-1 downto 0);
 clk, reset, start: in std_logic;
 x3: inout std_logic_vector(m-1 downto 0);
 y3: out std_logic_vector(m-1 downto 0);
 done: out std_logic
 );
end component;

component Point_double_v2 is
port(
 x1, y1: in std_logic_vector(m-1 downto 0);
 clk, reset, start: in std_logic;
 x3: inout std_logic_vector(m-1 downto 0);
 y3: out std_logic_vector(m-1 downto 0);
 done: out std_logic
 );
end component;

signal xR0, yR0, xR1, yR1, xR2, yR2, xR3, yR3,
    next_xR0, next_yR0, next_xR1, next_yR1, next_xR2, next_yR2, next_xR3, next_yR3:
std_logic_vector(m-1 downto 0);

signal int_k0, int_k1, int_s: std_logic_vector(m-1 downto 0);

signal add_in_x1, add_in_y1, add_in_x2, add_in_y2, dou_in_x, dou_in_y,
    add_out_x, add_out_y, dou_out_x, dou_out_y: std_logic_vector(m-1 downto 0);
```

```
signal ce_R0, ce_R1, ce_R2, ce_R3, start_add, start_dou, add_done, dou_done, k_m_minus_1_0,
k_m_minus_1_1, s_m_minus_1,
     reset_counter, count_down, last_step: std_logic;

signal step_number: std_logic_vector(logm-1 downto 0);

signal sel_R0, sel_R1, sel_R2, sel_R3, sel_in_dou, sel_in_add: std_logic_vector(1 downto 0);

subtype states is natural range 0 to 36;

signal current_state: states;

begin

with sel_R0 select next_xR0 <= add_out_x when "01", dou_out_x when "10", xR when others;
with sel_R0 select next_yR0 <= add_out_y when "01", dou_out_y when "10", yR when others;
with sel_R1 select next_xR1 <= add_out_x when "01", dou_out_x when "10", x_pplusr when others;
with sel_R1 select next_yR1 <= add_out_y when "01", dou_out_y when "10", y_pplusr when others;
with sel_R2 select next_xR2 <= add_out_x when "01", dou_out_x when "10", xR when others;
with sel_R2 select next_yR2 <= add_out_y when "01", dou_out_y when "10", y_minus_R when
others;
with sel_R3 select next_xR3 <= add_out_x when "01", dou_out_x when "10", x_pminusr when
others;
with sel_R3 select next_yR3 <= add_out_y when "01", dou_out_y when "10", y_pminusr when
others;
with sel_in_add select add_in_x1 <= xR0 when "00", xR2 when others;
with sel_in_add select add_in_y1 <= yR0 when "00", yR2 when others;
with sel_in_add select add_in_x2 <= xR1 when "00", xR0 when "11", xR3 when others;
with sel_in_add select add_in_y2 <= yR1 when "00", yR0 when "11", yR3 when others;
with sel_in_dou select dou_in_x <= xR0 when "00", xR1 when "01", xR2 when "10", xR3 when
others;-- "11"
with sel_in_dou select dou_in_y <= yR0 when "00", yR1 when "01", yR2 when "10", yR3 when
others;

register_R0: process(clk)
begin
if clk' event and clk = '1' then
if ce_R0 = '1' then xR0 <= next_xR0; yR0 <= next_yR0; end if;
end if;
end process;

register_R1: process(clk)
begin
if clk' event and clk = '1' then
if ce_R1 = '1' then xR1 <= next_xR1; yR1 <= next_yR1; end if;
end if;
end process;

register_R2: process(clk)
```

```
begin
if clk' event and clk = '1' then
if ce_R2 = '1' then xR2 <= next_xR2; yR2 <= next_yR2; end if;
end if;
end process;

register_R3: process(clk)
begin
if clk' event and clk = '1' then
if ce_R3 = '1' then xR3 <= next_xR3; yR3 <= next_yR3; end if;
end if;
end process;

add: Point_add_v2 port map(add_in_x1, add_in_y1, add_in_x2, add_in_y2, clk, reset, start_add,
add_out_x, add_out_y, add_done);

double: Point_double_v2 port map(dou_in_x, dou_in_y, clk, reset, start_dou, dou_out_x, dou_out_y,
dou_done);

with step_number select last_step <= '1' when "00000000", '0' when others;

shift_register_k0: process(clk)
begin
if clk'event and clk = '1' then
if reset_counter = '1' then int_k0 <= k0;
elsif count_down = '1' then
for i in m-1 downto 1 loop int_k0(i) <= int_k0(i-1); end loop;
int_k0(0) <= '0';
end if;
end if;
end process;

k_m_minus_1_0 <= int_k0(m-1);


shift_register_k1: process(clk)
begin
if clk'event and clk = '1' then
if reset_counter = '1' then int_k1 <= k1;
elsif count_down = '1' then
for i in m-1 downto 1 loop int_k1(i) <= int_k1(i-1); end loop;
int_k1(0) <= '0';
end if;
end if;
end process;

k_m_minus_1_1 <= int_k1(m-1);

shift_register_s: process(clk)
```

```
begin
if clk'event and clk = '1' then
if reset_counter = '1' then int_s <= s;
elsif count_down = '1' then
for i in m-1 downto 1 loop int_s(i) <= int_s(i-1); end loop;
int_s(0) <= '0';
end if;
end if;
end process;

s_m_minus_1 <= int_s(m-1);


output_xQ: process(clk, last_step)
begin
if clk' event and clk = '1' then
if last_step = '1' then
for i in 0 to m-1  loop
   xQ(i) <= xR0(i);
            yQ(i) <= yR0(i);
end loop;
end if;
end if;
end process;

end process;
end;
```

VITA AUCTORIS

Che Chen was born in 1989 in P.R.China. He received his Bachelor's Degree from Faculty of Science in Wuhan University of Technology in 2011. He received his Master of Engineering degree from the Department of Electrical and Computer Engineering at University of Windsor in 2013. He is currently a candidate for the Master of Applied Science Degree in the Department of Electrical and Computer Engineering at University of Windsor and hopes to graduate in Fall, 2017