4-14-2017

Efficient scalar multiplication against side channel attacks using new number representation

Yue Huang
University of Windsor

Efficient scalar multiplication against side channel attacks using new number representation

by

Yue Huang

A Thesis

Submitted to the Faculty of Graduate Studies

through Electrical and Computer Engineering

in Partial Fulfilment of the Requirements for

the Degree of Master of Applied Science

at the University of Windsor

Windsor, Ontario, Canada

2017

Efficient scalar multiplication against Side Channel Attacks using new number representation

by

Yue Huang

APPROVED BY:

Dr. Arunita Jaekel

School of Computer Science

Dr. Kemal Tepe

Department of Electrical and Computer Engineering

Dr. Mitra Mirhassani, Co-Supervisor

Department of Electrical and Computer Engineering

Dr. H. Wu, Supervisor

Department of Electrical and Computer Engineering

Jan. 21, 2017

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 (ECC) is probably the most popular public key systems nowadays. The classic algorithm for computation of elliptic curve scalar multiplication is Doubling-and-Add. However, it has been shown vulnerable to simple power analysis, which is a type of side channel attacks (SCAs). Among different types of attacks, SCAs are becoming the most important and practical threat to elliptic curve computation. Although Montgomery power ladder (MPL) has shown to be a good choice for scalar multiplication against simple power analysis, it is still subject to some advanced SCAs such like differential power analysis. In this thesis, a new number representation is firstly proposed, then several scalar multiplication algorithms using this new number system are presented. It has also been shown that the proposed algorithms outperform or comparable to the best of existing similar algorithms in terms of against side channel attacks and computational efficiency. Finally we extend both the new number system and the corresponding scalar multiplication algorithms to high radix cases.

DEDICATION

To my family:

Father: Wei Huang

Mother: Yufeng Cong

Sister: Chao Huang

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude and appreciation to everyone who helped me during my graduate study.

First of all, I am deeply grateful to my supervisor Dr. Huapeng Wu, who guided me during my study for the master degree. He is always patient with me. He guided me throughout the writing of this thesis. His broad knowledge have been of great value. In addition, I would like to show my gratitude to my co-supervisor Dr. Mitra Mirhassani, who gave me a lot of valuable advices and guidances during my research. Also, I want to thank my committee members, Dr. Kemal Tepe and Dr. Arunita Jaekel, for their attendance and advices in my seminar and defense.

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

Finally, I wish to extend my gratitude to everyone at UWindsor's Faculty of Electrical and Computer Engineering for their efforts during my study in the M.A.Sc. Program. Moreover, I gratefully thank for the financial support from University of Windsor and my supervisor Dr. Huapeng Wu.

Yue Huang

TABLE OF CONTENTS

DECLARATION OF ORIGINALITY iii

ABSTRACT iv

DEDICATION v

ACKNOWLEDGEMENTS vi

LIST OF TABLES x

LIST OF FIGURES xi

LIST OF ALGORITHMS xii

LIST OF ACRONYMS xiv

1 Introduction 1

1.1 Motivation 1

1.2 Main Contributions 2

1.3 Thesis Organization 3

2 Public Key Cryptography and Side Channel Attacks 4

2.1 Asymmetric Cryptography 4

2.1.1 Elliptic curve cryptosystem and scalar multiplication 4

2.2 Side Channel Attacks 5

2.2.1 Timing attack 6

2.2.2 Powering analysis 6

2.2.3 Fault attack 7

2.2.4 Other side channel attacks 7

3 Overview of Existing Works **9**

3.1 Doubling-and-Add Always Algorithms 9

3.1.1 Doubling Attack 10

3.1.2 C Safe Error Attack 11

3.2 Montgomery Powering Ladder 11

3.2.1 Relative Doubling Attack 12

3.2.2 M Safe Error Attack 13

3.2.3 Comparative Power Analysis 16

3.2.4 Park's Fault Attack 16

3.3 Follow up Countermeasures on Exponent and Message Masking 18

3.3.1 Exponent Splitting 18

3.3.2 High Orders Attack against Exponent Splitting 19

3.3.3 Blinded Fault Resistant Exponentiation 19

3.3.4 Template Attack against Masked MPL 20

3.4 Sequence Masking 21

3.5 Combined Sequence Masking and Exponent Splitting algorithm proposed by He 21

3.6 Joye's m-ary Algorithms 21

3.6.1 Joye's Binary Algorithms 24

4 Proposed Elevated Binary Number System (EBNS) **27**

4.1 Elevated Binary Number System Property 27

4.1.1 Convert between EBNS and Binary Number System 27

4.1.2 Conversion between EBNS and Decimal Number System 28

4.2 Elevated High Radix Number System 29

5 Proposed Algorithms using Elevated Number System **31**

5.1 Binary Representation 31

5.2 High Radix Representation 35

5.3 Security Analysis and Comparison 38

5.3.1 Binary Representation 38

5.3.2 High Radix Representation 44

5.4 Complexity Analysis and Comparison 46

5.4.1 Binary Representation 46

5.4.2 High Radix Representation 48

6 Conclusions and Possible Future Works **54**

6.1 Conclusions 54

6.2 Possible Future Works 54

REFERENCES **56**

VITA AUCTORIS **60**

LIST OF TABLES

2.1 Side channels and Corresponding Side channel attacks 6

3.1 Computations of M^e and $(M^2)^e$ using Montgomery Powering Ladder 14

3.2 Computation of modular exponentiation when a fault is injected in the squaring operation 18

3.3 Computation of modular exponentiation when a fault is injected in the squaring operation 18

5.1 Computations of kP and $2(kP)$ using Proposed Alg 5.1 40

5.2 Security against SCAs among Left-to-Right algorithms (Binary Representation) 41

5.3 Computation of Alg 5.3 when a fault is injected in the addition operation . . 43

5.4 Computation of Alg 5.6 when a fault is injected in the addition operation . . 43

5.5 Security against SCAs among Right-to-Left algorithms (Binary Representation) 44

5.6 Comparison against SCAs among High Radix algorithms 46

5.7 Complexity comparison among left-to-right algorithms (Binary Representation) 46

5.8 Complexity comparison among right-to-left algorithms (Binary Representation) 47

5.9 Complexity comparison among high radix algorithms 48

5.10 Complexity comparison among high radix algorithms 49

LIST OF FIGURES

3.1 Example of comparative power attack against MPL 17

4.1 Architecture of converting EBNS into Binary Number System 28

4.2 Architecture of converting Binary Number System into EBNS 29

5.1 Example of comparative power attack against Alg 5.1 39

LIST OF ALGORITHMS

3.1 Doubling-and-Add algorithm (left-to-right) 9

3.2 Doubling-and-Add algorithm (right-to-left) 9

3.3 Square-and-Multiply Always algorithm (left-to-right) 10

3.4 Montgomery Powering Ladder 12

3.5 Rewrite square-and-multiply algorithm 15

3.6 Masked Montgomery Powering Ladder 20

3.7 Joye's m-ary left-to-right algorithm (RSA) 22

3.8 Joye's m-ary left-to-right algorithm (ECC) 23

3.9 Joye's m-ary right-to-left algorithm (RSA) 24

3.10 Joye's m-ary right-to-left algorithm (ECC) 24

3.11 Joye's binary left-to-right algorithm (RSA) 25

3.12 Joye's binary left-to-right algorithm (ECC) 25

3.13 Joye's binary right-to-left algorithm (RSA) 26

3.14 Joye's binary right-to-left algorithm (ECC) 26

4.1 Convert EBNS into Binary Number 27

4.2 Convert Binary Number into EBNS 28

4.3 Convert Decimal Number into EBNS 29

5.1 Scalar multiplication using EBNS (left-to-right) 31

5.2 Scalar multiplication using EBNS (right-to-left) version 1 32

5.3 Scalar multiplication using EBNS (right-to-left) version 2 33

5.4 Scalar multiplication using EBNS (right-to-left)version 3 33

5.5 Scalar multiplication using EBNS (right-to-left) version 4 34

5.6 Scalar multiplication using EBNS (right-to-left) version 5 35

5.7 Scalar multiplication using Elevated High Radix Number (left-to-right) . . 35

5.8 Scalar multiplication using Elevated High Radix Number (right-to-left) version 1 36

5.9 Scalar multiplication using Elevated High Radix Number (right-to-left) . . 37
5.10 Scalar multiplication using Elevated High Radix Number (right-to-left) version 2 38
5.11 Scalar multiplication using Elevated High Radix Number (right-to-left) version 3 38

LIST OF ACRONYMS

DPA Differential Power Analysis

EBNS Elevated Binary System

ECC Elliptic Curve Cryptosystem

ECDLP Elliptic Curve Discrete Logarithm Problem

EM Electromagnetic

MPL Montgomery Power Ladder

SCAs Side channel attacks

SPA Simple Power Analysis

1 Introduction

1.1 Motivation

The Internet has an enormous impact on many aspects of our daily life. One of the great advantage of Internet is that it makes data communication much easier. Accordingly, cyber security gains people's attentions more than ever. Among a variety of technologies to provide cyber security, cryptography is probably the most important and effective one. Cryptography can provide many security services, such like confidentiality and authentication. For instance, public key cryptography is well known for providing digital signature and solving key distribution problems. Elliptic curve cryptography (ECC) is considered the most popular public key systems nowadays. The most demanded computation in ECC is scalar multiplication.

However, there exist many kinds of attacks which can potentially compromise the security of the cryptosystems. Besides the brute force attack, mathematical attacks and protocol level attacks [1], side channel attacks (SCAs) [2] has become probably the most dangerous threat to cryptosystems. SCAs utilize the information leaked from the physical implementation of a cryptosystem, rather than the weaknesses in the algorithms and protocols. The leakage information includes timing information, power consumption, electromagnetic radiation and sounds etc. By collecting and analysing these side channel information, an attacker is able to launch the corresponding SCAs.

The classic algorithms to compute scalar multiplication is doubling-and-add algorithm. Due to the unbalanced number of operations for different value of scalar bits, doubling-and-add algorithm is vulnerable to the most basic SCA, simple power analysis (SPA) [3]. SPA involves that an attacker directly observes the power spectrum of a device on which the cryptography algorithm is being performed. As an improved scalar multiplication algorithm, MPL can resist to SPA since it has balanced number of operation all the time, thus corresponding power consumption reveals no useful information about the algorithm.

Recently, a few sophisticated SCAs have been proposed to against MPL, such as relative doubling attack [4], comparative power analysis [5] and Park's fault attack [6]. For example, comparative power analysis uses two related inputs to generate collisions for MPL. A more recent algorithm described in [7] by Joye can successfully resist to the SCAs proposed in [4] [5] but still shows weakness in resisting Park's fault attack [6]. In this thesis, we proposed new scalar multiplication algorithms which are shown more advantageous over or comparable to the best of existing similar algorithms, in terms of resistance to SCAs and computational efficiency.

1.2 Main Contributions

The main contributions of this thesis are summarized as follows:

- A new binary number system, called Elevated binary number system (EBNS), is proposed. Conversions between this new number system and the conventional binary number is discussed. An extension of EBNS to high radix system is also presented.

- New left-to-right scalar multiplication algorithm using EBNS is proposed. This algorithm shows advantage in terms of computation efficiency while maintains the same security strength against SCAs compared to the best existing work. We also propose five right-to-left algorithms using EBNS. Compared to the best existing work, all proposed 5 algorithms have a advantage of not having a constraint condition. One of the proposed right-to-left algorithm is able to resist Park's fault attack [6] while all other existing work can not.

- We propose one left-to-right algorithm using elevated high radix number system. Compared to the best existing work described in [7], the main superiority of this algorithm is to reduce complexity while not sacrificing the security strength. We also propose three right-to-left algorithms. One of them can successfully resist to Park's fault attack while the existing work can not.

1.3 Thesis Organization

An organization of the rest of this thesis is as follows. Chapter 2 presents the background of (ECC). A brief overview of existing works is given in Chapter 3. Chapter 4 proposes a new number system, Elevated Binary Number System (EBNS). In Chapter 5, we propose several scalar multiplication algorithms using new number system including binary cases and high radix cases. Their comparison with existing work in terms of complexity and security strength against side channel attacks are presented in this chapter. In Chapter 6, the conclusion and possible future works are given.

2 Public Key Cryptography and Side Channel Attacks

2.1 Asymmetric Cryptography

Asymmetric cryptography, also known as public key cryptography, uses public and private keys to encrypt and decrypt data. One key in the pair which can be shared with public is called public key. The other key in the pair is kept secret named private key. Either of the keys can be used to encrypt a message, while the opposite key is used for decryption.

Public key cryptography is addressed to provide two main security services: key distribution, which is used to exchange keys without having to trust a third party, and digital signature, which is used to verify the intactness of a message from a claimed sender. There are several public key cryptosystems which are frequently used currently, such as RSA and ECC. Compared with symmetrical key cryptosystems, public key cryptosystems usually have higher computational complexity while provides unique security services, which makes them indispensable in cyber security.

2.1.1 Elliptic curve cryptosystem and scalar multiplication

RSA and ECC are the most widely used and best known asymmetric cryptosystems. The invention of RSA was in later 1970s at MIT, by Ron Rivest, Adi Shamir and Len Adleman [8]. And such cryptosystem is named after the first digit of their last names. The core operation of RSA algorithm is modular exponentiation. Different from RSA, ECC has not been patented. The suggestion of using elliptic curve in cryptography is first published in [9] in 1985. Scalar multiplication is the main computation in ECC algorithm. Scalar multiplication is very similar to modular exponentiation. On one hand, they both work with cyclic group. On the other hand, since they share the similar computation structure, they are subjects to similar SCAs. Because of the similarity, in the following of the paper, we mainly focus on analysing scalar multiplication algorithms and the analysis can be easily extended to modular exponentiation algorithms.

Assume Alice wants to use ECC for communication. She needs to set up an ECC system. It includes parameters of p, a, b, P, n, h, Q, d. First, she selects a prime p and elliptic curve E with (a, b) over $GF(p)$.

$$E : y^2 = x^3 + ax + b, \text{ where } a \text{ and } b \in GF(p).$$

Second, she counts the points on E and let it be $\sharp E(GF(p))$. Then she selects a point P of the maximal order n on E, which $h = \sharp E(GF(p))/n$. For public-private key pair (Q, d), Alice needs to choose a secret number $d \in [1, n-1]$. Then computes scalar multiplication $Q = dP$. Q is Alice's public key, d is the private key.

In a scenario, Bob wants to send an encrypted message to Alice using ECC. Assume m is the original message. He should choose a random number r, and computes $C_1 = m + rQ$, $C_2 = rP$. Then sends C_1 and C_2 to Alice. Upon receiving the message, Alice needs to computes $C_1 - dC_2 = m$ to get the original message m.

From the computations mentioned above, we can see the core operations of ECC encryption and decryption is point addition and point doubling based on elliptic curve. In order to illustrate how to compute point addition and point doubling, we firstly assume an elliptic curve:

$$E : y^2 = x^3 + ax + b, \text{ where } a, b \in GF(p).$$

Then let $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$ be the two points on the curve.

Assume $P_3(x_3, y_3) = P_1(x_1, y_1) + P_2(x_2, y_1) \neq O$. Then we can get

$$x_3 = \lambda^2 - x_1 - x_2; \; y_3 = \lambda(x_1 - x_3) - y_1;$$

where $\lambda = \frac{y_2 - y_1}{x_2 - x_1}$ if $P_1 \neq P_2$ and $\lambda = \frac{3x_1^2 + a}{2y_1}$ if $P_1 = P_2$.

2.2 Side Channel Attacks

The side channel attacks we consider are a class of physical attacks in which an attacker tries to exploit physical information leakages such as timing information, power consumption, or electromagnetic radiation [2].

Depends on what kind of leakage of system information SCAs relies on, we can cate-

Table 2.1: Side channels and Corresponding Side channel attacks

Side Channels Information	Side Channel Attacks
Power Consumptions	Simple Power Analysis, Differential Power Analysis, Comparative Power Analysis etc.
Timing information	Timing attack
Faults response	Safe-error Attacks
Other SCAs: Electromagnetic Radiation, Sounds etc.	EM Attacks, Acoustic cryptanalysis

gorized it into several types. The common types of leakage information and corresponding side channel attacks are shown in Table 2.1.

2.2.1 Timing attack

When the running time of a cryptographic device is not constant, this time may leak information about the secret parameters involved, so that careful timing measurement and analysis may allow an attacker to recover the system's secret key. This idea was first appeared in the scientific literature in 1996 [10].

Timing attack is practical in many cases. For example, in 2003, Boneh and Brumley demonstrated a network-based timing attack on an OpenSSL-based web server running on a machine in a local network. The attack was able to successfully recover a server private key [11].

2.2.2 Powering analysis

Power analysis involves an attacker studies the power consumption of a cryptographic hardware device (such as a smart card). It can be roughly divided in two types: simple power analysis (SPA) and differential power analysis (DPA) [2].

In simple power analysis (SPA), an attacker directly observes a device's power spectrum. It is known that the amount of power consumed by the device varies depending on

the data operated on and the instructions performed during different parts of an algorithm's execution [3].

Differential power analysis (DPA) is a more advanced form of power analysis which can allow an attacker to compute the intermediate values within cryptographic computations by statistically analysing data collected from multiple cryptographic operations [12].

2.2.3 Fault attack

Fault attack is a well known cryptanalysis method in which an attacker purposely induces certain types of fault into a cryptographic device. Based on the correctness of outputs from the device due to the injection of faults, an adversary is able to obtain certain knowledge of the secret key which is embedded in the device [6]. These so-called fault attacks were first described against public key schemes by Boneh, DeMillo and Lipton [13].

As illustration of the attack scheme, one of fault based attack mentioned by Sung Ming Yen and Marc Joye is explained as follows. In [14] and [15], they describe the attack as this: By timely inducing a fault during the execution of an instruction, an attacker may deduce whether the targeted instruction is redundant. If the final result is correct then the instruction is indeed redundant (or dummy operation [15]). If not, the instruction is effective. This knowledge may then be used to obtain one or more bits of an exponent. Such attacks are referred to as safe-error attack. Since safe error attack is able to check the effectiveness of each operation, it is dangerous to have dummy operations in cryptographic algorithms. More fault attacks can be found in [6] [16] [17].

2.2.4 Other side channel attacks

Electromagnetic attack (EM) is a side channel with a long history of rumors and leaks associated with its use for espionage. It is well known that defence organizations across the world are paranoid about limiting electromagnetic emanations from their equipment and facilities and conduct research on EM attacks and defences in total secrecy [3].

In the public domain, the significance of the EM side channel was first demonstrated by van Eck in 1985 [18]. He showed that EM emanations from computer monitors could be captured from a distance and used to reconstruct the information being displayed.

Acoustic cryptanalysis is a type of side channel attacks which exploits sounds emitted by computers or machines. Modern acoustic cryptanalysis mostly focuses on the sounds produced by computer keyboards and internal computer components, but historically it has also been applied to impact printers and electromechanical cipher machines [19].

One obvious countermeasure is to use sound dampening equipment, such as a silent keyboard. It can be a keyboard made of rubber, or a keyboard based on a touchscreen or touchstream technologies. Recently, virtual keyboards have appeared that can be projected on a flat surface or in the air [20].

3 Overview of Existing Works

3.1 Doubling-and-Add Always Algorithms

The most classic algorithms to compute scalar multiplication is Doubling-and-Add algorithms, which are presented as Alg. 3.1 and Alg.3.2. The first one computes from the most significant bit, while the other one starts from the least significant bit.

Algorithm 3.1 Doubling-and-Add algorithm (left-to-right)

Input: Point $P \in E$, $k = (k_{h-1}k_{h-2}...k_1k_0)_2, \quad k_i \in \{0,1\}$
Output: $R = kP \in E$
1: $x = P$;
2: **for** $i = h-2$ down to 0 **do**
3: $x = 2x$;
4: **if** $k_i = 1$ **then**
5: $x = x + P$;
6: **end if**
7: **end for**
8: The final value of x is $x = kP$.

We can see from Alg 3.1 and Alg 3.2, if k_i equals to 1, there are two operations performed, which are one doubling and one addition. If $k_i = 0$, only one doubling operation is computed. Since the number of operation varies from different value of key bits, the power spectrum of performing this algorithm can certainly reveal the value of each key bit. So doubling-and-add algorithms are obviously vulnerable to SPA.

Algorithm 3.2 Doubling-and-Add algorithm (right-to-left)

Input: Point $P \in E$, $k = (k_{h-1}k_{h-2}...k_1k_0)_2, \quad k_i \in \{0,1\}$
Output: $R = kP \in E$
1: $x = O$;
2: $y = P$;
3: **for** $i = 0$ down to $h-1$ **do**
4: **if** $k_i = 1$ **then**
5: $x = x + y$;
6: **end if**
7: $y = 2y$;
8: **end for**
9: The final value of x is $x = kP$.

Square-and-multiply always algorithm as an improvement to against SPA was invented by J. Coron in 1999 [21]. The algorithm which computes scalar multiplication has also been developed. The left-to-right version of algorithm is shown as Alg 3.3. Dummy operation is introduced in the algorithm to balance the number of operations for different value of key bits. Though this algorithm resists to SPA, it's still vulnerable to newer attacks such as doubling attack [22] and C safe error attack [15].

Algorithm 3.3 Square-and-Multiply Always algorithm (left-to-right)

Input: Point $P \in E$, $k = (k_{h-1}k_{h-2}...k_1k_0)_2, \quad k_i \in \{0,1\}$
Output: $R = kP \in E$

```
1: x = O;
2: y = O;
3: for i = h − 1 down to 0 do
4:     x = 2x;
5:     y = x + P;
6:     if k_i = 1 then
7:         x = y;
8:     end if
9: end for
10: The final value of x is x = kP.
```

3.1.1 Doubling Attack

Doubling attack was proposed in 2003 [22]. It's so named since it is based on the doubling operation in the scalar multiplication. Consider left-to-right version of doubling-and-add algorithm as an example, the main idea of this attack is to compute kP and $k(2P)$ at the same time and compare the power spectrum of their doubling operations. As a result, we can observe that if and only if the secret key bit k_i equals to 0, the doubling operation in iteration $i+1$ for compute kP is the same as in iteration i for compute $k(2P)$. So if a collision is detected, the information of $k_i = 0$ can be obtained. This attack is also effective to against square-and-multiply algorithms, in which the attacker chooses to compute m^e and $(m^2)^e$ instead.

3.1.2 C Safe Error Attack

Another attack square-to-multiply always algorithm vulnerable to is C safe error attack. It was proposed by S. Yen in 2002 [15]. It specifically targets algorithms with dummy operations. The error an attacker injects to is a temporary random computational error. The attack is so named as C safe error attack. We use Alg 3.3 as an example to illustrate how this attack works in detail. If the attacker injects an error into the dummy operation $y = x + P$, then this error will be ignored if the secret bit $k_i = 0$. This is because the faulty result y will not participate into the next operation $x = y$. As a result, the algorithm is correct and the secret bit $k_i = 0$ can be extracted. Otherwise, $k_i = 1$ is obtained.

Even doubling attack and C safe error attack have been proven effectively against several algorithms, they can be prevented by using a newer algorithm, Montgomery Powering Ladder.

3.2 Montgomery Powering Ladder

Montgomery Powering Ladder (MPL) was initially invented as an improvement of square-and-multiply algorithm towards SPA in 2003 [23]. As shown in Algorithm 3.4, it always performs a multiplication followed with a squaring. Consequently, there is no difference in power consumption regarding to compute different value of bits. Also, there is no dummy operation in the algorithm, the error induced by C safe error attack would always cause a faulty result. Thus, no useful information can be obtained to tell the value of key bits. Moreover, the original doubling attack can not tell the value of key bits either.

Even MPL can resist to many SCAs, it is still vulnerable to some advanced side channel attacks, such as relative doubling attack [24], M safe error attack [14], comparative power analysis [5] and Park's fault attack [6].

Algorithm 3.4 Montgomery Powering Ladder

Input: $M, N, e = (e_{n-1}e_{n-2}...e_1e_0)_2, \quad e_i \in \{0,1\}$
Output: $C = M^e \bmod N$
1: $R_0 = 1; R_1 = M;$
2: **for** $i = n-1$ down to 0 **do**
3: **if** $e_i = 0$ **then**
4: $R_1 = R_0 \times R_1; \quad R_0 = {R_0}^2;$
5: **else**
6: $R_0 = R_0 \times R_1; \quad R_1 = {R_1}^2;$
7: **end if**
8: **end for**
9: The final value of C is $C = M^e \bmod N$.

3.2.1 Relative Doubling Attack

Relative doubling attack was proposed in 2006 to specifically attack Montgomery power ladder [24]. The assumption of this attack is basically the same as what is in doubling attack. The difference is that the relative doubling attack aims to derive the knowledge of whether or not the key bits equal to each other but not the value of key bits directly. The relative doubling attack is based on the following observations of MPL:

Let $\sum_{i=0}^{t-1} e_i 2^i$ be the binary expansion of exponent e. Defining $L_j = \sum_{i=j}^{t-1} e_i 2^{i-j}$ and $H_j = L_j + 1$. Then we have

$$L_j = 2L_{j+1} + e_j = L_{j+1} + H_{j+1} + e_j - 1 = 2H_{j+1} + e_j - 2$$

Based on the above observation, we can obtain

$$L_j = 2L_{j+1}, H_j = L_{j+1} + H_{j+1}, \text{if } e_j = 0;$$

$$L_j = L_{j+1} + H_{j+1}, H_j = 2H_{j+1}, \text{if } e_j = 1;$$

Recall from the MPL algorithm, the operations of step 4 and step 6 are designed to be as follows:

Equation 1: $(R_1 = M^{H_i}, R_0 = M^{L_i}) = (M^{L_{i+1}} * M^{H_{i+1}}, ({M^{L_{i+1}}}^2)$ if $e_i = 0$;

Equation 2: $(R_0 = M^{L_i}, R_1 = M^{H_i} = (M^{L_{i+1}} * M^{H_{i+1}}, ({M^{H_{i+1}}}^2)$ if $e_i = 1$;

It's easy to observe two facts:

Fact 1. Given $e_i = 0$, then we have $L_i = 2L_{i+1}$.

Fact 2. Given $e_i = 1$, then we have $H_i = 2H_{i+1}$.

From Equation 1, we understand that if $e_i = e_{i-1} = 0$ then both

$R_0 = M^{L_i}{}^2$: step 4 of iteration $i-1$ when evaluating M^e.

$R_0 = (M^2)^{L_{i+1}}$: step 4 of iteration i when evaluating $(M^2)^e$.

will perform the same computation because of $L_i = 2L_{i+1}$. Due to this observation of collision on computation, a new doubling-like attack can be implemented to obtain the knowledge of $e_i = e_{i-1} = 0$.

From Equation 2, we also observe that if $e_i = e_{i-1} = 1$, then both

$R_1 = M^{H_i}{}^2$: step 6 of iteration $i-1$ when evaluating M^e.

$R_1 = (M^2)^{H_{i+1}}$: step 6 of iteration i when evaluating $(M^2)^e$.

will perform the same computation because of $H_i = 2H_{i+1}$. This observation of collision on computation leads to the knowledge of $e_i = e_{i-1} = 1$.

It's easy to understand that the collision of two computations will not reveal the value of the operand. The only knowledge obtained is $e_i = e_{i-1}$, if a collision is detected. On the other hand, $e_i \neq e_{i-1}$, if no collision is detected.

For example, we assume the private exponent e to be $75 = (1001011)_2$ and the two related input to be M^e and $(M^2)^e$ respectively. Table 3.2 provides the details of computing M^e and $(M^2)^e$ using MPL. Given $e_0 = 1$, then step 6 of MPL algorithm will be calculated. If we observe a collision of comparing doubling operation in iteration 0 for compute M^e and in iteration 1 for compute $(M^2)^e$, then we can obtain the knowledge of $e_0 = e_1 = 1$. If no collision is detected, $e_0 = 1$ is obtained. The idea of this attack is also applied to MPL algorithm computes scalar multiplication. Instead of choosing M^e and $(M^2)^e$, the two related input data should be chosen as kP and $2(kP)$.

3.2.2 M Safe Error Attack

M safe error attack is the first published safe error based attack, which was proposed in 2000 [14]. In order to explain how the attack works in detail, we need to rewrite the

Table 3.1: Computations of M^e and $(M^2)^e$ using Montgomery Powering Ladder

i	e_i	Process of M^e	Process of $(M^2)^e$
6	1	$R_0 = 1 \times M$; $R_1 = M^2$	$R_0 = 1 \times M^2$; $R_1 = (M^2)^2$
5	0	$R_1 = M^2 \times M$; $R_0 = M^2$	$R_1 = M^4 \times M^2$; $R_0 = (M^2)^2$
4	0	$R_1 = M^3 \times M^2$; $R_0 = (M^2)^2$	$R_1 = M^6 \times M^4$; $R_0 = (M^4)^2$
3	1	$R_0 = M^4 \times M^5$; $R_1 = (M^5)^2$	$R_0 = M^8 \times M^{10}$; $R_1 = (M^{10})^2$
2	0	$R_1 = M^{10} \times M^9$; $R_0 = (M^9)^2$	$R_1 = M^{20} \times M^{18}$; $R_0 = (M^{18})^2$
1	1	$R_0 = M^{18} \times M^{19}$; $R_1 = (M^{19})^2$	$R_0 = M^{36} \times M^{38}$; $R_1 = (M^{38})^2$
0	1	$R_0 = M^{37} \times M^{38}$; $R_1 = (M^{38})^2$	$R_0 = M^{74} \times M^{76}$; $R_1 = (M^{76})^2$
Returen		$R_0 = M^{75}$	$R_0 = M^{150}$

classic square-and-multiply algorithm first. The commonly used algorithm for computing $M^e \mod N$ where the exponent e is expressed in the binary form as $e = \sum_{i=0}^{n-1} e_i 2^i$. Suppose that the modular multiplication $R = AB \mod N$ has to be preformed. Let $\sum_{i=0}^{n-1} a_i 2^i$ be the binary expansion of A. Then, A can be recoded in radix 2^t as $\sum_{j=0}^{m-1} A_j (2^t)^j$ instead. Hence we can rewrite the product of A and B. Algorithm 3.5 shows the rewrite version of square-and-multiply algorithm [14].

Algorithm 3.5 Rewrite square-and-multiply algorithm

Input: $M, N, e = (e_{n-1}, e_{n-2} \dots e_0)_2$
Output: $A = M^e \bmod N$

1: $A = 1; B = M;$
2: **for** $i = 0$ to $n-1$ **do**
3: **if** $e_i = 1$ **then**
4: $R = 0;$
5: **for** j from $m-1$ down to 0 **do**
6: $R = (R2^t + A_j B) \bmod N;$
7: **end for**
8: $A = R;$
9: **end if**
10: $R = 0;$
11: **for** j from $m-1$ down to 0 **do**
12: $R = (R2^t + B_j B) \bmod N;$
13: **end for**
14: $B = R;$
15: **end for**

The main idea of M safe error can be understood as follows: We can see from algorithm 3.5, if one or several errors are introduced into the more significant bit positions of register A, after restoring the result R into A (step 8), no error will be detected if the faulty bits belong to words A_j are no loner required. For example, we assume that during iteration i, an error is injected into word A_k. In the scenario that $e_i = 1$ and the error is induced when counter j of step 6 is less than k, then the faulty A_k will not affect the final result of R. Eventually, after step 8, the faulty A will be cleared, Such kind of temporary error is called a safe error. But in the scenario of $e_i = 0$, the final result will be incorrect. Thus, we can obtain the value of e_i [14] [2].

M safe error attack is effectively against MPL too. Recall from Alg 3.4, suppose that R_1 is assigned as the multiplier. If $e_i = 0$, two operations $R_1 = R_0 \times R_1$ and $R_0 = (R_0)^2$ are performed. An error injected into the higher part of R_1 is an M safe error, since the error in register R_1 will be cleared after the operation $R_1 = R_0 \times R_1$. The error does not affect the final result of the algorithm. If $e_i = 1$, the error injected into R_1 will cause a faulty result. Based on these two distinct results, an attacker can recover the value of bit e_i. It's the same

case of setting R_0 as the multiplier.

3.2.3 Comparative Power Analysis

Comparative power analysis is a power analysis technique which was proposed in 2000 [5]. It can attack multiple algorithms including square-and-multiply algorithms, square-and-multiply always algorithms, Sliding window methods and MPL.

It's similar to doubling attack only more powerful, the basic idea is also to input a chosen messages to generate collisions. Instead of choosing M^e and $(M^e)^2$, The message pair could be selected as $(M^e)^\alpha$ and $(M^e)^\beta$. We can see, if $2\alpha = \beta$, it can be seen as the doubling attack. It's the same case for MPL algorithm that computes scalar multiplication. The message pair can be chose as αkP and βkP.

Fig.3.1 shows an example of comparative power analysis applied to MPL. Suppose the attacker already knows the first four bits of the exponent e, which are $(1100)_2$. Assuming that $e_{k-5} = 1$, we use the value of $R_1 (= Y^{13})$ as the input of the squaring operation. Since $e_{k-2} = 1$, the message pair $Y^{13} = Z^2$ is then selected to meet the condition. Compare the power traces of the two shadow parts, if the results are identical, it means the assumption is correct. otherwise, the assumption is wrong, which means $e_{k-5} = 0$.

3.2.4 Park's Fault Attack

MPL is also vulnerable to Park's fault attack which was proposed in 2009 [6]. The main idea of this attack is to inject a fault into one loop of the algorithm and check the correctness of the next loop's result. The attacker is able to obtain the relationship of two adjacent secret bits.

Take MPL as an example, the basic assumption is that the attacker can insert a fault during the exponentiation operation and measure the power consumption traces. We also assume that the attacker knows the power traces of multiplication $R_0 \times R_1 \bmod N$ and squaring $R_0 \times R_0 \bmod N$ of every iteration loop i. There are two scenarios to inject a fault.

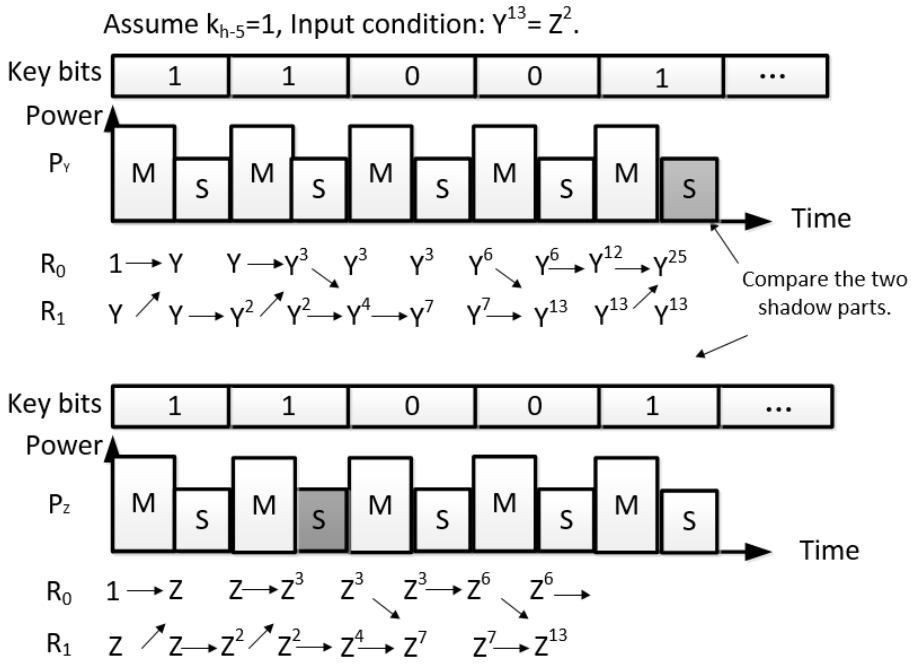

Fig. 3.1: Example of comparative power attack against MPL

In the first scenario, we assume that an attacker injects a fault during the multiplication computation R_1 or $R_0 = R_0 \times R_1$ of the loop $i+1$. Recall from algorithm 3.4, if $e_{i+1} = 0/1$, then the intermediate value R_1/R_0 becomes a faulty one. The output of this scenario is shown as Table 3.2. From the table, we can see if bit $e_{i+1} = e_i$, the two power traces of squaring operation in loop i will be the same. However, when the two key bits are different, the attacker is able to observe difference between two power traces. As a result, the attacker can obtain the knowledge of whether or not e_{i+1} equals to e_i. More specifically, if an attacker who knows the value of e_{i+1}, he can obtain the value of next key bit e_i by observing the power differences in loop i.

In the second scenario, we assume that an attacker injects a fault during the squaring computation R_0^2 or R_1^2 of loop $i+1$. If $e_{i+1} = 0/1$, then intermediate value R_0/R_1 becomes a faulty one. This fault model is independent of the multiplication step of the loop $i+1$, and has the same assumption of fault type on the multiplication operation. The output of this scenario is shown as Table 3.3. As we can see, if key bit e_{i+1} differs from e_i, the two

Table 3.2: Computation of modular exponentiation when a fault is injected in the squaring operation

Secret key bit		loop $i+1$		loop i	
e_{i+1}	e_i	Multiply	Squaring	Multiply	Squaring
0	0	A fault is injected, generate faulty value	Normal	Error	Normal
0	1				Error
1	0				Error
1	1				Normal

Table 3.3: Computation of modular exponentiation when a fault is injected in the squaring operation

Secret key bit		loop $i+1$		loop i	
e_{i+1}	e_i	Multiply	Squaring	Multiply	Squaring
0	0	Don't care	A fault is injected, generate faulty value	Error	Error
0	1				Normal
1	0				Normal
1	1				Error

power traces in the squaring operation in loop i will be the same. However, when key bits e_{i+1} and e_i are the same, the erroneous processing within squaring operation in the loop i will generate some power differences between the two power traces. So an attacker who knows e_{i+1} can also derive the next secret bit e_i by observing the power spectrum in the loop i.

3.3 Follow up Countermeasures on Exponent and Message Masking

3.3.1 Exponent Splitting

The idea of data splitting was first abstracted in [25]. And in [26], the idea was used specifically on exponent. For modular exponentiation, the idea of splitting is to pick a random r (smaller than e) and split e as $r' = e - r$. Then the modular exponentiation can be computed as $m^e = m^{r+(e-r)} = m^r \times m^{e-r}$. The recovery process can be easily performed as

$S = S_r \times S_{r'} = M^{r+r'} \bmod N = M^e \bmod N$. The weakness of this method is it doubles the computation load and can be broke by high order attack [27].

3.3.2 High Orders Attack against Exponent Splitting

High Orders Attack was proposed in [27] by Frederic Muller and Frederic Valette. They discovered some statistical properties of the exponent splitting. Then they used their discovery to mount side channel attacks.

The properties can be understood as this: $C_i = C_i \oplus r_i \oplus r'_i$ is always satisfied, where C_i is the carry bit in i-th iteration and r_i, r'_i refer to the two split elements of exponent e. P_i is defined as the probability for the case of $C_i = 0$ and $P_r(r_i, r'_i)$ as the probability for bracket case, we could have the probability transactions. From it, the bit-level probabilities of step i can be derived from previous steps.

With this property in hand, an attacker can apply it with side channel attacks such as fault attacks, safe error attacks and address-bit attacks to learn the secret exponent. Then exponent splitting countermeasure is no longer secure.

3.3.3 Blinded Fault Resistant Exponentiation

Blinded Fault Resistant Exponentiation is also known as Masked Montgomery Powering Ladder (Masked MPL). It was proposed in [26] by G. Fumaroli and D. Vigilant in 2009. It is a message masking technique based on MPL. It's shown as Algorithm 3.6. At the beginning of the algorithm, two registers R_0 and R_1 are blinded by multiply a random picked number r. Then all the intermediate values of R_0 and R_1 are masked by the element $r^{2^{n-i}}$.

The register R_2 is initialized with the anti-mask r^{-1} and such anti-mask is also updating during each iteration process. Multiply R_0 and R_2 at the end of the algorithm will maintain the correctness of the result. In addition, in order to thwart potential fault attack and exponent or loop counter disturbance, an on-the-fly checksum function is performed to satisfy

such purpose.

Other than resist to SPA, Masked MPL also successfully withstand differential power analysis (DPA) and fault attacks. Unless the random number r is revealed, or it is a weak mask, Masked MPL is considered secure. And thanks to the Checksum function's participation, most fault attack cannot pass the very last sum checking. Failure in such checking will cause the calculated results wiped. However this attack is vulnerable to template attack, which is able to reveal the secret mask.

Algorithm 3.6 Masked Montgomery Powering Ladder

Input: $M, e = (e_{n-1}e_{n-2}...e_1e_0)$; init(CKS)
Output: $C = M^e$

Pick random number r
$R_0 = r; R_1 = rM; R_2 = r^{-1}$;
for $i = n-1$ down to 0 **do**
  if $e_i = 0$ **then**
    $R_1 = R_0 \times R_1; \quad R_0 = {R_0}^2; \quad R_2 = {R_2}^2$
    update (CKS, e_i);
  else
    $R_0 = R_0 \times R_1; \quad R_1 = {R_1}^2; \quad R_2 = {R_2}^2$
    update (CKS, e_i);
  end if
  $R_2 = R_2 \oplus CKS \oplus CKS_{ref}$;
end for
Return $C = R_0 \times R_2$.

3.3.4 Template Attack against Masked MPL

C. Herbst and M. Medwed proposed the template attacks in [28]. It has been proved that it is effective to attack Masked Montgomery Ladder via guessing the value of random mask [28].

Template attack can be seen as a advanced case of SPA. First, the attacker tries to obtain the Hamming weights from obtained power traces. In this stage, DPA, probability model multivariate normal distribution (MVN) and maximum likelihood (ML) decision rule are used. Second, the attacker will filter out impossible operand values to recover an operand of a multiplication or a squaring. It involves two steps. In the first sieving step, wrong

candidates are filtered out. In the second step, the correct bit will be selected. Hence, the mask will be revealed.

3.4 Sequence Masking

Sequence masking is another effective countermeasure to resist certain side channel attacks. It usually changes the procedure of exponentiation methods. One example is Square-and-Multiply-always method [21]. There is few work about sequence masking. since it's difficult to change the computation sequence in an algorithm. carelessly modifying may damage its correctness. Also, simply adding dummy operation may make the algorithm more vulnerable. Nevertheless, sequence masking is a possible countermeasure of preventing SCAs.

3.5 Combined Sequence Masking and Exponent Splitting algorithm proposed by He

A highly secure algorithm which combines sequence masking and exponent splitting technique together was proposed by He in 2013. In [29], it shows that despite its high complexity, the algorithm can successfully resist to all the side channel attacks mentioned above.

3.6 Joye's m-ary Algorithms

There is another algorithm called highly regular m-ary powering ladders [7], which can be seen as m-ary generalizations of Montgomery ladder. It was originally invented to compute modular exponentiation in RSA. It can be easily extended to perform scalar multiplication in ECC. Similar to MPL, this algorithm always repeat the same instructions in the same order, without inserting dummy operations. It's available in any radix m and in any compute direction (left-to-right or right-to-left).

Joye's m-ary algorithms derive highly regular exponentiation algorithms by considering

a representation of $e-1$ instead of the original exponent e. The m-ary algorithms can be understood as follows:

Let $e = \sum_{i=0}^{l-1} e_i m^i$ denotes the m-ary representation of exponent e. Then

$$e = (e_{l-1} - 1)m^{l-1} + (\sum_{i=0}^{l-2}(d_i + m - 1)m^i) + 1;$$

The equation above can be rewritten as

$$e - 1 = \sum_{i=0}^{l-1} e_i^* m^i;$$

where $e = e_i^* = e_i + m - 1$ for $0 \leqslant i \leqslant l-2$; $e = e_i^* = e_{l-1} - 1$ for $i = l-1$.

For the left-to-right algorithm which computes modular exponentiation in RSA is shown as Algorithm 3.7, it makes use of an register A, initialized to $g^{e_{l-1}^*}$. At each iteration of the main loop, the register A is raised to the power of m and then always multiplied by $g^{e_i^*}$. At the end of the main loop, the register is multiplied by g to get the correct result. The corresponding algorithm that computes scalar multiplication is shown as Alg 3.8.

Algorithm 3.7 Joye's m-ary left-to-right algorithm (RSA)

Input: Point $g \in G$, $d = \sum_{i=0}^{h-1} d_i m^i, \quad d_i \in \{0, 1, ..., m-1\}$
Output: g^d

1: **for** $i = 1$ to m **do**
2: $\quad R[i] = g^{m+i-2}$;
3: **end for**
4: $A = g^{d_{h-1}-1}$
5: **for** $i = h-2$ down to 0 **do**
6: $\quad A = A^m$;
7: $\quad A = A \times R[1 + d_i]$;
8: **end for**
9: $A = A \times g$;
10: The final value is $A = g^d$.

Algorithm 3.8 Joye's m-ary left-to-right algorithm (ECC)

Input: Point $P \in E$, $k = (k_{h-1}k_{h-2}...k_1k_0)_2, \quad k_i \in \{0,1...m-1\}$
Output: kP

1: **for** $i = 1$ to m **do**
2: $\quad R[i] = (m+i-2)P$;
3: **end for**
4: $A = (k_{h-1}-1)P$
5: **for** $i = h-2$ down to 0 **do**
6: $\quad A = mA$;
7: $\quad A = A + R[1+k_i]$;
8: **end for**
9: $A = A + P$;
10: The final value is $A = kP$.

For the right-to-left algorithm, we have

$$g^{e-1} = ((g^m)^{l-1})^{d^*_{l-1}} \times \prod_{j=1}^{m-1}(L^*_j)^{m+j-2};$$

where $L^*_j = \prod_{0 \leqslant i \leqslant l-2, e^*_i = j} g^{m^i}$.

The right-to-left algorithm uses m registers, $R[1]...R[m]$ to keep track of the value of L^*_j and an accumulator that keeps track of the successive value of g^{m^i}. Same case as left-to-right algorithm, at the end, one multiplication is needed as the correct step to maintain the correctness of the algorithm. The m-ary right-to-left version algorithm is shown as Algorithm 3.9. Corresponding algorithm performs scalar multiplication is shown as Alg 3.10.

Algorithm 3.9 Joye's m-ary right-to-left algorithm (RSA)

Input: Point $g \in G$, $d = \sum_{i=0}^{h-1} d_i m^i, \quad d_i \in \{0,1,...,m-1\}$
Output: g^d
1: **for** $i = 1$ to m **do**
2: $R[i] = 1_G$;
3: **end for**
4: $A = g$;
5: **for** $i = 0$ to $h-2$ **do**
6: $R[1+d_i] = R[1+d_i] \times A$;
7: $A = A^m$;
8: **end for**
9: $A = A^{d_{h-1}-1} \times \prod_{i=1}^{m} R[i]^{m+i-2}$;
10: $A = A \times g$;
11: The final value is $A = g^d$.

Algorithm 3.10 Joye's m-ary right-to-left algorithm (ECC)

Input: Point $P \in E$, $k = (k_{h-1}k_{h-2}...k_1k_0)_2, \quad k_i \in \{0,1...m-1\}$
Output: kP
1: **for** $i = 1$ to m **do**
2: $R[i] = O$;
3: **end for**
4: $A = P$;
5: **for** $i = 0$ to $h-2$ **do**
6: $R[1+k_i] = R[1+k_i] + A$;
7: $A = mA$;
8: **end for**
9: $A = (k_{h-1}-1)A + \sum_{i=1}^{m}(m+i-2)R[i]$;
10: $A = A + P$;
11: The final value is $A = kP$.

3.6.1 Joye's Binary Algorithms

If $m = 2$, from high radix algorithms, we obtain the binary cases, in which the exponent $e \in \{0,1\}$. Then $e - 1 = \sum_{i=0}^{l-2} e_i^* 2^i$ with $e_i^* = e_i + 1$. Also, e_{l-1} is set to be 1. For the left-to-right version, the algorithm computes modular exponentiation is shown as Alg 3.11 and the corresponding algorithm computes scalar multiplication is presented as Alg 3.12. As seen in Alg 3.11, register A is initialized to $R[e_{h-2}+1]$ and the loop starts at index $l-3$. In each loop, there is always a squaring operation followed by one multiplication. Also there

is a correction step which includes one multiplication at the end of the algorithm.

Algorithm 3.11 Joye's binary left-to-right algorithm (RSA)

Input: Point $g \in G$, $d = \sum_{i=0}^{h-1} d_i 2^i, \quad d_i \in \{0,1\}$
Output: g^d
1: $R[1] = g$;
2: $R[2] = R[1]^2$;
3: $A = R[d_{h-2}+1]$
4: **for** $i = h-3$ down to 0 **do**
5: $A = A^2$;
6: $A = A \times R[1+d_i]$;
7: **end for**
8: $A = A \times R[1]$;
9: The final value is $A = g^d$.

Algorithm 3.12 Joye's binary left-to-right algorithm (ECC)

Input: Point $P \in E$, $k = (k_{h-1}k_{h-2}...k_1k_0)_2, \quad k_i \in \{0,1\}, \quad k_{h-1} = 1$
Output: $kP \in E$
1: $x = (k_{h-2}+1)P$;
2: $y = 2P$;
3: **for** $i = h-3$ down to 0 **do**
4: **if** $k_i = 0$ **then**
5: $x = 2x; \quad x = x+P$;
6: **else**
7: $x = 2x; \quad x = x+y$;
8: **end if**
9: **end for**
10: $x = x+P$;
11: The final value is $x = kP$.

For right-to-left algorithm, the most significant bit is also set to be 1. In order to prevent the final correction step, $R[1]$ is initialized to be g^{e_0} and $R[2]$ to g. Then the loop can start at index 1 instead of 0. At the end of the algorithm, one squaring and one multiplication are needed to correct the final result. The algorithm computes modular exponentiation is shown as Alg 3.13 and the corresponding algorithm performs scalar multiplication is presented as Alg 3.14.

Algorithm 3.13 Joye's binary right-to-left algorithm (RSA)

Input: Point $g \in G$, $d = \sum_{i=0}^{h-1} d_i 2^i, \quad d_i \in \{0,1\}$
Output: g^d

1: $R[1] = g^{d_0}$;
2: $R[2] = g$;
3: $A = R[2]$
4: **for** $i = 1$ to $h-2$ **do**
5: $\quad A = A^2$;
6: $\quad R[1+d_i] = A \times R[1+d_i]$;
7: **end for**
8: $A = R[1] \times R[2]^2$;
9: The final value is $A = g^d$.

Algorithm 3.14 Joye's binary right-to-left algorithm (ECC)

Input: Point $P \in E$, $k = (k_{h-1}k_{h-2}...k_1k_0)_2, \quad k_i \in \{0,1\}, \quad k_{h-1} = 1$
Output: $kP \in E$

1: $x = k_0P$;
2: $y = P$;
3: $z = P$;
4: **for** $i = 1$ to $h-2$ **do**
5: $\quad$ **if** $k_i = 0$ **then**
6: $\quad\quad z = 2z; \quad x = x+z$;
7: $\quad$ **else**
8: $\quad\quad z = 2z; \quad y = y+z$;
9: $\quad$ **end if**
10: **end for**
11: $z = x+2y$;
12: The final value is $z = kP$.

4 Proposed Elevated Binary Number System (EBNS)

4.1 Elevated Binary Number System Property

For an integer in EBNS of length k (there are k digits in the representation),

$$X = (x_{k-1}x_{k-2}...x_1x_0)_2; \text{ where } x_i \in \{1,2\}$$

The maximal representable value is $2^{k+1}-2$ and the minimal representable value can be calculated as 2^k-1. So, the representation range of k digits for this new number system is $\left[2^k-1, 2^{k+1}-2\right]$. By contrast, the representation rang of k digits for binary number is $\left[2^{k-1}, 2^k-1\right]$. So, for the same value, EBNS representation needs one bit less than binary number system.

For example, if we want to represent decimal number 27 in this new number system. We can write it as $(2211)_2$.

4.1.1 Convert between EBNS and Binary Number System

We also present the conversion algorithms between binary number system and EBNS. The algorithms are shown as Algorithm 4.1 and 4.2. The main difference between EBNS and binary number system is their digit set. We can consider that for EBNS, we use digit 1 to replace binary digit 0 and replace binary digit 1 by 2.

Algorithm 4.1 Convert EBNS into Binary Number

Input: $x = (x_{k-1}x_{k-2}...x_1x_0)_2, \quad x_i \in \{1,2\}$
Output: $y = (y_ky_{k-1}y_{k-2}...y_1y_0)_2 \quad y_i \in \{0,1\}$
1: $x_i --; i = 0,1,...,k-1;$
2: $y := (1x_{k-1}...x_0) - 1;$
3: The final value is $y = (y_k...y_1y_0)_2 \quad y_i \in \{0,1\}$

We can see from Alg 4.1 and Alg 4.2, when converting EBNS into binary number, only one binary subtraction is needed. when converting binary number into EBNS, a binary addition operation is performed. The architecture of these two conversions are shown as Figure 4.1 and Figure 4.2. They are both low-cost conversion. Because of the different

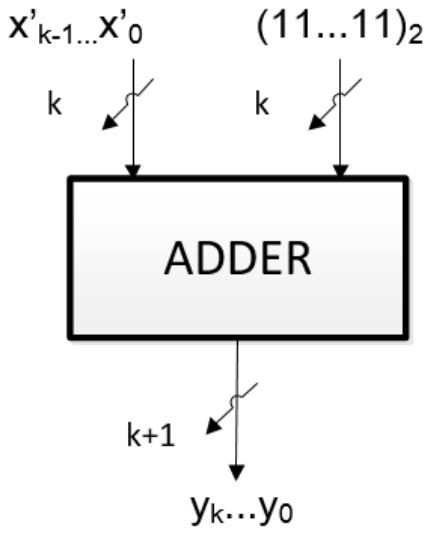

Fig. 4.1: Architecture of converting EBNS into Binary Number System

digit sets, for the same value, EBNS representation is always one bit less than binary representation. There is one unique situation, which is all the bits are 1. It's the same case for binary number and Elevated binary number. But it has no practical use. We can ignore this situation.

Algorithm 4.2 Convert Binary Number into EBNS

Input: $x = (x_{k-1}x_{k-2}...x_1x_0)_2, \quad x_i \in \{0,1\}, \quad x_{k-1} \neq 0$
Output: $y = (...y_1y_0)_2, \quad y_i \in \{1,2\}$
1: $y = (x_{k-2}...x_1x_0) + 1;$
2: $y_i++; i = 0,1,...,k-2;$
3: The final value is $y = (...y_1y_0)_2 \quad y_i \in \{1,2\}$

4.1.2 Conversion between EBNS and Decimal Number System

It's easy to convert EBNS into decimal number system, we just need to calculate the value of the elevated binary number. For converting decimal number into EBNS, we propose Algorithm 4.3.

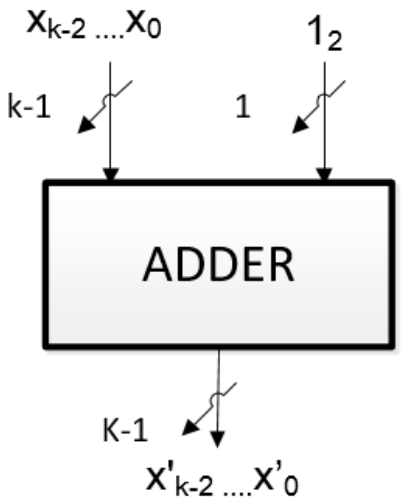

Fig. 4.2: Architecture of converting Binary Number System into EBNS

Algorithm 4.3 Convert Decimal Number into EBNS

Input: x in decimal representation
Output: $y = (y_{k-1}y_{k-2}...y_1y_0)_2, \quad y_i \in \{1,2\}$

1: $i = 0$;
2: **while** $x \neq 0$ **do**
3: $y = x \bmod 2$;
4: **if** $y_i = 0$ **then**
5: $y_i = 2; \quad x = x/2 - 1$;
6: **else**
7: $x = (x-1)/2$;
8: **end if**
9: **end while**,
10: $i = i + 1$;
11: The final value is $y = (y_{k-1}y_{k-2}...y_1y_0)_2$

4.2 Elevated High Radix Number System

We further extend our elevated binary number system idea into high radix m, we propose Elevated High Radix Number System. For an integer in Elevated High Radix Number

System of length k (there are k digits in the representation),

$$X = (x_{k-1}x_{k-2}...x_1x_0)_m; \text{ where } x_i \in \{1, 2, \cdots, m\}$$

The maximal representable value is $\frac{m(1-m^k)}{1-m}$. and the minimal representable value can be calculated as $\frac{(1-m^k)}{1-m}$. So, the representation range of k digits for this new number system is $\left[\frac{(1-m^k)}{1-m}, \frac{m(1-m^k)}{1-m}\right]$.

5 Proposed Algorithms using Elevated Number System

In this section, several algorithms compute scalar multiplication using our proposed new number systems are proposed. Algorithms using EBNS are firstly presented. Then we extend the algorithms to high radix cases. We also show the proposed algorithms outperform or comparable to the best of existing similar algorithms in terms of against side channel attacks and computational efficiency.

5.1 Binary Representation

For left-to-right version of algorithm, we propose Algorithm 5.1. The correctness of this algorithm is easy to prove. The core operation can be wrote as $\{x = 2x;\ x = x + k_i P\}$. After $h-1$ loops of computation, we can get the result of $x = 2^{h-1}x + 2^{h-2}k_{h-2}P + 2^{h-3}k_{h-3}P + ... + k_0 P$. Since x is initiated as $k_{h-1}P$, final result $x = kP$ is obtained.

For each loop, the core computation includes one doubling and one addition no matter k equals to 1 or 2. Apparently, with this balanced number of operation for different value of key bits, this algorithm can resist to SPA.

Algorithm 5.1 Scalar multiplication using EBNS (left-to-right)

Input: Point $P \in E$, $k = (k_{h-1}k_{h-2}...k_1k_0)_2, \quad k_i \in \{1,2\}$
Output: $kP \in E$
1: $x = k_{h-1}P$;
2: $y = 2P$;
3: **for** $i = h-2$ down to 0 **do**
4: **if** $k_i = 1$ **then**
5: $x = 2x; \quad x = x + P$;
6: **else**
7: $x = 2x; \quad x = x + y$;
8: **end if**
9: **end for**
10: The final value is $x = kP$.

For right-to-left algorithms, due to the different initiations steps and correction steps, we propose five algorithms. The first one is shown as Algorithm 5.2. The main computation

can be wrote as $\{z = 2y;\ x = x + (2 - k_i)y;\ x = x + (k_i - 1)y;\ y = z;\}$. After computing from loop 1 to loop $h-1$. the result of x is $k_0P + 2^1k_1P + 2^2k_2P + ... + 2^{h-1}k_{h-1}P$. Since x is initiated as O, we can obtain the final result $x = kP$. As seen from Alg 5.2, at the end of each loop, register z needs to be assigned to y. The main computation in every loop also includes one doubling and one addition. Apparently, Alg 5.2 resist to SPA.

Algorithm 5.2 Scalar multiplication using EBNS (right-to-left) version 1

Input: Point $P \in E$, $k = (k_{h-1}k_{h-2}...k_1k_0)_2, \quad k_i \in \{1,2\}$
Output: $kP \in E$

1: $x = k_0P$;
2: $y = 2P$;
3: **for** $i = 1$ to $h-1$ **do**
4: **if** $k_i = 1$ **then**
5: $z = 2y; \quad x = x + y;$
6: **else**
7: $z = 2y; \quad x = x + z\ ;$
8: **end if**
9: $y = z;$
10: **end for**
11: The final value is $x = kP$.

The second algorithm of computing from right to left is illustrated as Algorithm 5.3. The main computation can be wrote as $\{x = x + (2 - k_i)z;\ y = y + (k_i - 1)z;\ z = 2z;\}$. So after the $h-1$ loops of computation, the result of x is $k_0P + 2^1k_1P + 2^2k_2P + ... + 2^{h-1}k_{h-1}P$. Since x is initiated as O, the final result $x = kP$ is obtained. We can see from Alg 5.3, in each loop, there is one addition and one doubling, which can be computed in parallel. There are three registers x,y and z which are initialized as O,O and P. A correction step $z = x + 2y$ is needed at the end of the algorithm.

Algorithm 5.3 Scalar multiplication using EBNS (right-to-left) version 2

Input: Point $P \in E$, $k = (k_{h-1}k_{h-2}...k_1k_0)_2, \quad k_i \in \{1,2\}$
Output: $kP \in E$
1: $x = O$;
2: $y = O$;
3: $z = P$;
4: **for** $i = 0$ to $h-1$ **do**
5: **if** $k_i = 1$ **then**
6: $x = x+z; \quad z = 2z$;
7: **else**
8: $y = y+z; \quad z = 2z$;
9: **end if**
10: **end for**
11: $z = x+2y$;
12: The final value is $z = kP$.

The third right-to-left algorithm is shown as Algorithm 5.4. The core computation is the same as in Alg 5.3. It can also be calculated in parallel. Differ from Algorithm 5.3, the correctness step at the end of Alg 5.4 is $z = k_{h-1}z+x+2y$. There are $h-1$ loops in total. If the most significant bit (MSb) is 1, the last correction step includes one doubling and two additions. Otherwise, it contains two doubling and two additions.

Algorithm 5.4 Scalar multiplication using EBNS (right-to-left)version 3

Input: Point $P \in E$, $k = (k_{h-1}k_{h-2}...k_1k_0)_2, \quad k_i \in \{1,2\}$
Output: $kP \in E$
1: $x = O$;
2: $y = O$;
3: $z = P$;
4: **for** $i = 0$ to $h-2$ **do**
5: **if** $k_i = 1$ **then**
6: $x = x+z; \quad z = 2z$;
7: **else**
8: $y = y+z; \quad z = 2z$;
9: **end if**
10: **end for**
11: $z = k_{h-1}z+x+2y$;
12: The final value is $z = kP$.

Algorithm 5.5 is the forth right-to-left algorithm. It's also similar to Alg 5.3, the main computation is the same. The difference lie on the initialization step and correction step.

In Alg 5.5, we initialize register x to k_0P and register z to $2P$. The correction step is represented as $z = k_{h-1}z + x + 2y$. There are two cases for correction step. If $k_{h-1} = 1$, the correction step includes two additions and one doubling. Otherwise, it contains two additions and two doubling. There are $h-2$ loops involved.

Algorithm 5.5 Scalar multiplication using EBNS (right-to-left) version 4

Input: Point $P \in E$, $k = (k_{h-1}k_{h-2}...k_1k_0)_2, \quad k_i \in \{1,2\}$
Output: $kP \in E$

1: $x = k_0P$;
2: $y = O$;
3: $z = 2P$;
4: **for** $i = 1$ to $h-2$ **do**
5: **if** $k_i = 1$ **then**
6: $x = x + z; \quad z = 2z;$
7: **else**
8: $y = y + z; \quad z = 2z;$
9: **end if**
10: $z = k_{h-1}z + x + 2y;$
11: **end for**
12: The final value is $z = kP$.

The fifth right-to-left algorithm is shown as Algorithm 5.6. Register x, y and z are initialized as k_0P, O and P. The correction step at the end of the algorithm includes one doubling operation and one addition operation. The main computation can be wrote as $\{z = 2z;\ x = x + (2 - k_i)y;\ x = x + (k_i - 1)y;\ y = z;\}$. So after the last correction step, we can obtain the final result $x = kP$.

Algorithm 5.6 Scalar multiplication using EBNS (right-to-left) version 5

Input: Point $P \in E$, $k = (k_{h-1}k_{h-2}...k_1k_0)_2, \quad k_i \in \{1,2\}$
Output: $kP \in E$
1: $x = k_0P$;
2: $y = O$;
3: $z = P$;
4: **for** $i = 1$ to $h-1$ **do**
5: **if** $k_i = 1$ **then**
6: $z = 2z; \quad x = x+z$;
7: **else**
8: $z = 2z; \quad y = y+z$;
9: **end if**
10: $z = x+2y$;
11: **end for**
12: The final value is $z = kP$.

5.2 High Radix Representation

After propose algorithms compute scalar multiplication using EBNS, we extend the algorithms to high radix cases. The left-to-right version of algorithm is shown as Algorithm 5.7. m represents any high radix. As illustrated in the Alg 5.7, we initialize $R[i]$ to iP for i from 1 to m and A to $k_{h-1}P$. The main operations for every loop is $A = mA$; and $A = A + R[k_i]$. There is no correction step needed in the algorithm.

Algorithm 5.7 Scalar multiplication using Elevated High Radix Number (left-to-right)

Input: Point $P \in E$, $k = \sum_{i=0}^{h-1} k_i m^i, \quad k_i \in \{1,2,...m\}$
Output: $kP \in E$
1: **for** $i = 1$ to m **do**
2: $R[i] = iP$;
3: **end for**
4: $A = k_{h-1}P$;
5: **for** $i = h-2$ down to 0 **do**
6: $A = mA$;
7: $A = A + R[k_i]$;
8: **end for**
9: The final value is $kP = A$.

For right-to-left algorithms using high radix m, we propose three algorithms. Version 1

is shown as Alg. 5.8. For the initialization, $R[i]$ is set to be O_G for i from 1 to m and A to P. The main computation is $R[k_i] = R[k_i] + A$; and $A = mA$. The correction step at the end of the algorithm is $A = \sum_{i=1}^{m} iR[i]$. It can be rewrote as

$A = R[m]$;

For $i = m-1$ down to 1.

$R[i] = R[i] + R[i+1]$;

$A = A + R[i]$

End for

Alg 5.9 is the rewrote algorithm of Alg 5.8.

Algorithm 5.8 Scalar multiplication using Elevated High Radix Number (right-to-left) version 1

Input: Point $P \in E$, $k = \sum_{i=0}^{h-1} k_i m^i, \quad k_i \in \{1, 2, ...m\}$
Output: $kP \in E$

1: **for** $i = 1$ to m **do**
2: $R[i] = O_G$;
3: **end for**
4: $A = P$;
5: **for** $i = 0$ to $h-1$ **do**
6: $R[k_i] = R[k_i] + A$;
7: $A = mA$;
8: **end for**
9: $A = \sum_{i=1}^{m} iR[i]$;
10: The final value is $kP = A$.

The second version of right-to-left algorithm is shown as Alg. 5.10. It's similar to Alg. 5.8. The initialization steps and main computations are the same. The difference is that Alg 5.10 has one less loop and the correction step adds $k_{h-1}A$. The correction step can be rewrote as follows:

$A = (k_{h-1})A$;

$A = A + R[m]$;

For $i = m-1$ down to 1.

$R[i] = R[i] + R[i+1]$;

$A = A + R[i]$

End for

Algorithm 5.9 Scalar multiplication using Elevated High Radix Number (right-to-left)

Input: Point $P \in E$, $k = \sum_{i=0}^{h-1} k_i m^i, \quad k_i \in \{1, 2, ...m\}$
Output: $kP \in E$

1: **for** $i = 1$ to m **do**
2: $\quad R[i] = O_G$;
3: **end for**
4: $A = P$;
5: **for** $i = 0$ to $h-1$ **do**
6: $\quad R[k_i] = R[k_i] + A$;
7: $\quad A = mA$;
8: **end for**
9: $A = R[m]$;
10: **for** $i = m-1$ down to 1 **do**
11: $\quad R[i] = R[i] + R[i+1]$;
12: $\quad A = A + R[i]$;
13: **end for**
14: The final value is $kP = A$.

The third right-to-left algorithm is shown as Alg. 5.11. We can see from the algorithm, the main computation is $X = X + k_i Y$; $Y = mY$, which can be computed in parallel. $k_i Y$ is the multiple results of point P. If we calculate and restore all the multiple results of point P in advance, $k_i P$ does not need to be computed during the algorithm. The core operation $X = X + k_i Y$ can be seen as an addition, which is a great advantage in complexity reduce. No correction step is needed in this algorithm.

Algorithm 5.10 Scalar multiplication using Elevated High Radix Number (right-to-left) version 2

Input: Point $P \in E$, $k = \sum_{i=0}^{h-1} k_i m^i, \quad k_i \in \{1, 2, ...m\}$
Output: $kP \in E$
1: **for** $i = 1$ to m **do**
2: $\quad R[i] = O_G$;
3: **end for**
4: $A = P$;
5: **for** $i = 0$ to $h-2$ **do**
6: $\quad R[k_i] = R[k_i] + A$;
7: $\quad A = mA$;
8: **end for**
9: $A = k_{h-1}A + \sum_{i=1}^{m} iR[i]$;
10: The final value is $kP = A$.

Algorithm 5.11 Scalar multiplication using Elevated High Radix Number (right-to-left) version 3

Input: Point $P \in E$, $k = \sum_{i=0}^{h-1} k_i m^i, \quad k_i \in \{1, 2, ...m\}$
Output: $kP \in E$
1: $X = O_G$;
2: $Y = P$;
3: **for** $i = 0$ to $h-1$ **do**
4: $\quad X = X + k_i Y$;
5: $\quad Y = mY$;
6: **end for**
7: The final value is $kP = X$.

5.3 Security Analysis and Comparison

5.3.1 Binary Representation

In this subsection, we analyse our proposed binary algorithm's security strength against several side channel attacks and compared the results with existing works.

Since proposed Alg 5.1 has its feature of highly regular, which means it has balanced number of operation for different value of key bits. It can certainly resist to simple power analysis.

Doubling attack and comparative power analysis share the same principle of choosing related inputs with the purpose of generating collisions, which has been discussed in detail

in Section 3.2. However, for our proposed Alg 5.1, those two attacks have no use. In order to perform a doubling attack, the attacker needs to compute kP and $k(2P)$ at the same time and compare their power traces of doubling operation to observe a collision. In Alg 5.1, doubling operation is always performed before the addition operation no matter the secret key bit equals to 1 or 2. So no collision will be detected. Table 5.1 shows the results of computing scalar multiplication with input of $kP = 49P$ and $k(2P) = 98P$ using Alg 5.1. We can see from the table, there is no collision generated. As a result, Alg 5.1 can resist to doubling attack. It's the same case for launching comparative power analysis. Even we chose random input αkP and βkP, there is still no collisions generated since doubling operation is always performed before the addition operation in the loop. Figure 5.1 is an illustration for proposed algorithm against comparative power analysis, we assume the first four digit of k is $(1112)_2$ and the fifth digit is 1. Since $k_2 = 1$, we select message pair of $4A = B$. No matter k_4 equals to 1 or 2, the shadow parts are always the same. There is no way to obtain the value of k_4. So Alg 5.1 can resist to Comparative Power Analysis.

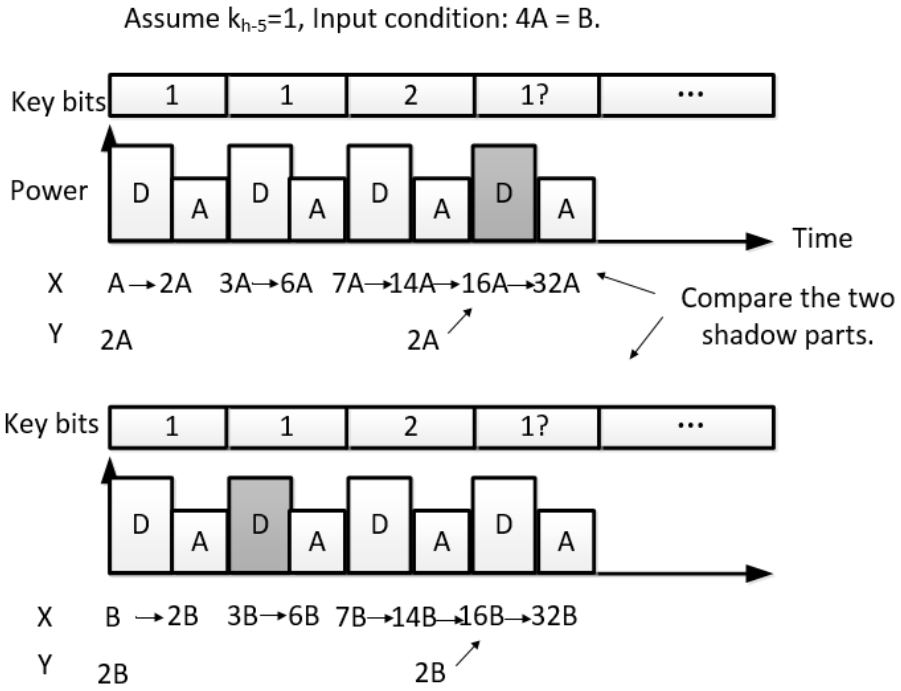

Fig. 5.1: Example of comparative power attack against Alg 5.1

Table 5.1: Computations of kP and $2(kP)$ using Proposed Alg 5.1

i	k_i	Process of kP	Process of $k(2P)$
4	2	$x = 2P$; $y = 2P$;	$x = 4P$; $y = 4P$;
3	1	$x = 2x = 4P$; $x = x + P = 5P$;	$x = 2x = 8P$; $x = x + 2P = 10; P$
2	1	$x = 2x = 10P$; $x = x + P = 11P$;	$x = 2x = 20P$; $x = x + 2P = 22P$;
1	2	$x = 2x = 22P$; $x = x + y = 24P$;	$x = 2x = 44P$; $x = x + y = 48P$;
0	1	$x = 2x = 48P$; $x = x + P = 49P$;	$x = 2x = 96P$; $x = x + 2P = 98P$;

C safe error Attack is specifically attack algorithms with dummy operations. Since proposed Alg 5.1 does not include dummy operation, this attack has no use toward Alg 5.1.

Proposed Alg 5.1 can also resist to M safe error attack. Recall from Section 3.2.2, the main idea of M safe error Attack is to inject an error into a register, if the register is restored afterwards, this attack can be seen as a safe error. Otherwise, it will result into a faulty result. For Alg 5.1, all the intermediate results are restored into register x, so the error injected into x will always cause a safe error. Then there is no way to distinguish the secret key bits from 1 to 2. So Alg 5.1 is M safe error prevented.

Park's fault attack is no threat to proposed Alg 5.1 either. Recall from Section 3.2.4, the main idea of this fault attack is to insert a fault during the computation of loop $i+1$ and check the correctness of results in loop i to distinguish the secret key bit. For Alg 5.1, all the intermediate results are stored into register x, so any fault injected in loop $i+1$ will always lead to a faulty result in loop i. Then there is no way to distinguish the different key bits.

High Order Attack is especially attack countermeasure of exponent splitting, so this attack is not applicable to Alg 5.1. Also, Template Attack is aim to attack message masking technique, it's also not applicable to proposed algorithm. For the rest proposed algorithms,

we will not analyse the security strength of proposed algorithms against these two attacks two attacks, since they are not applicable to all of them.

According to the above analysis, the comparison results of existing and proposed algorithm against Side Channel Attacks is presented in Table 5.2. As seen in Table 5.2, ✓ means the algorithm can resist to certain attack. while × indicates the algorithm is vulnerable to the certain attack. Our proposed Alg 5.1 has the same security strength compared to the best existing algorithm.

Table 5.2: Security against SCAs among Left-to-Right algorithms (Binary Representation)

	Existing algorithms				Proposed algorithm
	D-a-A Alg.3.1	D-a-A Always Alg.3.3	MPL Alg.3.4	Joye's Alg.3.12	Alg.5.1
Doubling attack	×	×	✓	✓	✓
Relative doubling attack	×	×	×	✓	✓
Comparative power analysis	×	×	×	✓	✓
Fault attack	×	×	×	✓	✓
M safe-error	×	×	×	✓	✓
C safe-error	×	×	✓	✓	✓

We propose five right-to-left algorithms using elevated binary representation. For every algorithm, we analyse its security strength against side channel attacks.

For Alg 5.2, since there is always a doubling operation followed by an addition in the main computation, doubling attack and comparative power analysis have no use towards this algorithm. C safe error attack is no use either, since there is no dummy operation involved in Alg 5.2. Recall from M safe error attack, if we inject an error into register x, the error will turn to be a safe error no matter $k_i = 1$ or $k_i = 2$. There is no way to tell the value of k_i. If the error is injected into y or z, the error will cause both faulty results for

$k_i = 1$ and $k_i = 2$. So there is still no way to distinguish the secret key bits. As a result, Alg 5.2 is M safe error attack prevented. Alg 5.2 is able to resist Park's fault attack too. Assume the attacker injects a fault into the doubling operation in loop i, the addition operation and the doubling operation in loop $i-1$ will both result in faulty outputs. There is no way to distinguish the secret key bit 1 from 2. If the fault is injected into the addition operation in loop i, the results of doubling operation in loop $i-1$ will all be correct. While the results of addition operation in loop i for both $k_i = 1$ and $k_i = 2$ are wrong. Still no way to distinguish the secret bits. So it's safe to say Alg 5.2 can resist to Park's fault attack.

Alg 5.3, Alg 5.4 and Alg 5.5 share a lot similarities, the main computations are the same, while the differences lie on initializations steps and correction steps. Due to their similarities, We analyse their security against side channel attacks together.

First, recall from these three algorithms, the doubling operation of main computation is $z = 2z$;, z will always double itself no matter k_i equals to 1 or 2. Also it has nothing to do with other registers like x and y. Then there is no way to generate collision by choosing different inputs. So doubling attack and comparative power analysis cannot attack these three algorithms. Second, C safe error attack has no use towards these three algorithms, since no dummy operation is involved in these three algorithms. Third, M safe error has no use towards these algorithms either, assume the attacker inject an error into register x. If $k_i = 1$, a safe error is generated. If $k_i = 2$, the error will not be able to affect the correctness of the result either. There is no way to tell the value of secret key bits. It's the same case for injecting error into register y and z. Unfortunately, these three algorithms are vulnerable to Park's fault attack. We use Alg 5.3 as an example, assume a fault is insert into addition operation in loop $i+1$. The computation results is shown in Table 5.3.

We can see from the table 5.4, if a fault is injected into addition operation in loop $i+1$, the doubling operation followed is normal. But for loop i, if k_{i+1} equals to k_i, the doubling operation is normal. Otherwise, the doubling operation will generate a faulty result. In this way, the attacker can obtain the knowledge of whether k_{i+1} equals to k_i or not.

Table 5.3: Computation of Alg 5.3 when a fault is injected in the addition operation

Secret key bit		loop $i+1$		loop i	
k_{i+1}	k_i	Doubling	Addition	Doubling	Addition
1	1	A fault is injected, generate faulty value	Normal	Error	Normal
1	2			Normal	
2	1			Normal	
2	2			Error	

Table 5.4: Computation of Alg 5.6 when a fault is injected in the addition operation

Secret key bit		loop $i+1$		loop i	
k_{i+1}	k_i	Doubling	Addition	Doubling	Addition
1	1	Don't care	A fault is injected, generate faulty value	Normal	Error
1	2				Normal
2	1				Normal
2	2				Error

Alg 5.6 is the fifth version of right-to-left algorithm. First, for the main computation, there is always a doubling operation followed by an addition operation. Also, z always double itself and has nothing to do with other registers. So, there is no way to generate collisions by choosing different inputs. Then, this algorithm can resist to doubling attack and comparative power analysis. Second, C safe error attack is still no use towards this algorithm. Any error injected into register x,y and z will all result in a safe error. Then there is no way to extract the value of secret key bit. So M safe error is no use either. Unfortunately, Park's fault attack can threat Alg 5.6. Assume a fault is injected into addition operation of loop $i+1$. The computation results is shown in Table 5.4. We can see from the table, if the addition operation in loop i generates correct results, we can obtain $k_{i+1} = k_i$. Otherwise, we can extract the information of k_i is differ from k_{i+1}.

According to the analysis above, the comparison results of existing and proposed right-to-left algorithms against side channel attacks is presented in Table 5.5. As seen in Table 5.6, ✓ means the algorithm can resist to certain attack. while $\times$ indicates the algorithm

is vulnerable to the certain attack. According to the table, Proposed Alg 5.11 has the best performance since it can resist to all the side channel attacks.

Table 5.5: Security against SCAs among Right-to-Left algorithms (Binary Representation)

	Existing algorithms		Proposed algorithms				
	D-a-A Alg.3.2	Joye's Alg.3.14	Alg.5.2	Alg.5.3	Alg.5.4	Alg.5.5	Alg.5.6
Doubling attack	×	✓	✓	✓	✓	✓	✓
Comparative power analysis	×	✓	✓	✓	✓	✓	✓
Fault attack	×	×	✓	×	×	×	×
M safe-error	×	✓	✓	✓	✓	✓	✓
C safe-error	✓	✓	✓	✓	✓	✓	✓

5.3.2 High Radix Representation

Alg 5.7 is the left-to-right version of compute scalar multiplication using high radix m. The security analysis of this algorithm is very similar to Alg 5.1. First, since the number of operation is the same for different value of secret key bit, the algorithm can certainly resist to simple power analysis. Second, the m-times operation $A = mA$ is always performed before the addition operation, there is no way to generate collisions by choosing different inputs. Then comparative powering analysis can not threat this algorithm. Third, since all the intermediate values are restored into A and there is no dummy operation involved, C safe error and M safe error has no use towards this algorithm either. Last, any fault injected into loop $i+1$ will always results in faulty output in loop i. So Alg 5.7 is Park's fault attack prevented.

Alg 5.8, Alg 5.10 and Alg 5.11 are the three versions of right-to-left algorithms. We analyse the security of Alg 5.8 and Alg 5.10 together, since most parts of these two algorithms are the same. First, the main computation always contains one addition and one

m-times operation for any key bit, so these two algorithms can certainly resist to SPA. Second, register A always m times itself during every loop, so comparative power analysis is not a threat. Third, for any value k_i, any error injected into $R[k_i]$ always results into a safe error and any error injected into A always cause a faulty output. So, there is no way to distinguish the secret key bits. M safe error is no threat either. Last, since the addition for main computation is $R[k_i] = R[k_i] + A$, there is no guarantee the same $R[k_i]$ will be computed in both loop $i+1$ and loop i. So, if any fault injected into loop $R[k_{i+1}]$ results into a faulty output in loop i, then we can obtain $R[k_{i+1}] = R[k_i]$. So Park's fault attack can successfully attack these two algorithms.

The main computation of Alg 5.11 is $X = X + k_i Y$; $Y = mY$, which can be seen as one addition and one m-times operation. Apparently, this algorithm is SPA prevented. Since register Y multiple itself during every loop, comparative power analysis is not a threat. No matter what the value of k_i is, any error injected into register X turns to be a safe error and any error injected into Y will cause faulty outputs. So, M safe error attack cannot threat this algorithm either. For Park's fault attack, if a fault is injected into X in loop $i+1$, it will cause addition operation in loop i a faulty result for all k_i. If the fault is injected into Y, the addition and m-times operation in loop i will both be wrong for every key bit. So, this algorithm can successfully against Park's fault attack.

Based on the analysis before, Table 5.6 shows the comparison results of security strength against side channel attacks between Joye's m-ary algorithms and proposed algorithms. From this table, we can see our proposed left-to-tight Alg 5.7 is as good as the existing Joye's m-ary algorithm. Proposed right-to-left Alg 5.11 can resist to all the side channel attacks mentioned above, while Joye's m-ary right-to-left algorithm is vulnerable to Park's fault attack.

Table 5.6: Comparison against SCAs among High Radix algorithms

	Existing algorithms		Proposed algorithms			
	Joye's m-ary Alg.3.8	Joye's m-ary Alg.3.10	Alg.5.7	Alg.5.8	Alg.5.10	Alg.5.11
Doubling attack	✓	✓	✓	✓	✓	✓
Comparative power analysis	✓	✓	✓	✓	✓	✓
Fault attack	✓	×	✓	×	×	✓
M safe-error	✓	✓	✓	✓	✓	✓
C safe-error	✓	✓	✓	✓	✓	✓

Table 5.7: Complexity comparison among left-to-right algorithms (Binary Representation)

	Existing algorithms				Proposed algorithm
	D-a-A (Alg.3.1)	D-a-A Always (Alg.3.3)	MPL (Alg.3.4)	Joye's (Alg.3.12)	Alg.5.1
# of loops	$L(k)-1$	$L(k)$	$L(k)$	$L(k)-2$	$L(k)-2$
Constraint	MSb=1	-	-	MSb=1	-
Correction step	-	-	-	One point add.	-
# of Addition	H(k)	$L(k)$	$L(k)$	$L(k)-1$	$L(k)-2$
# of Doubling	$L(k)-1$	$L(k)$	$L(k)$	$L(k)-2$	$L(k)-2$

5.4 Complexity Analysis and Comparison

5.4.1 Binary Representation

In order to compare the complexity of algorithms, we focus on 5 properties, which are number of loops, number of addition, number of doubling, algorithm constraint and correction step. For left-to-right version of algorithms, we compare the traditional doubling-and-add algorithm, doubling-and-add always algorithm, MPL, Joye's binary case and our proposed Alg. 5.1. Table 5.7 shows the comparison results.

$L(k)$ shown in table 5.7 is defined as $\lceil \log_2 k \rceil$. $H(k)$ indicated the Hamming weight of

Table 5.8: Complexity comparison among right-to-left algorithms (Binary Representation)

	Existing algorithms		Proposed algorithms		
	D-a-A (Alg.3.2)	Joye's (Alg.3.14)	Alg.5.2	Alg.5.3	Alg.5.6
No. of loops	$L(k)$	$L(k)-2$	$L(k)-2$	$L(k)-1$	$L(k)-2$
Constraint	-	MSb=1	-	-	-
Correction step	-	One Dou. One Add.	One Dou. One Add.	One Dou. One Add.	One Dou. One Add.
No. of Addition	$L(k)$	$L(k)-1$	$L(k)-2$	L(k)	$L(k)-1$
No. of Doubling	$L(k)$	$L(k)-1$	$L(k)-2$	$L(k)$	$L(k)-1$

k. *MSb* represents the most significant bit of secret key bit k.

We can see from Table 5.7, compared to existing works, our proposed Alg. 5.1 has the least number of loops, number of addition and number of doubling. Also, unlike Alg 3.9, proposed Alg.5.1 does not have a constraint of MSb=1 nor correction step. It is safe to say, proposed Alg.5.1 has the best complexity performance.

We also compare the right-to-left version of algorithms, which is shown as Table 5.8. Since Alg 5.3, Alg 5.4 and Alg 5.5 share a lot similarities. We only pick Alg 5.3 to compare with other algorithms since it has the best complexity performance. The correction step of Alg 5.4 and Alg 5.6 have two possibilities since the MSb k_{h-1} could be 1 or 2. If $k_{h-1} = 1$, the correction step includes one doubling and two addition. Otherwise, there are two doubling and two addition. Accordingly, the number of addition and doubling are variable.

We can see from Table 5.8, compared to the best existing work, Joye's algorithm, all proposed algorithms have an advantage of not having a constraint. Also, Alg 5.6 have the same performance on number of loops, addition and doubling as Joye's. Alg 5.2 has a better performance compared to Joye's algorithm since it has one less loop, addition and doubling.

Table 5.9: Complexity comparison among high radix algorithms

	Joye's algorithms		Proposed Algorithms			
	L2R (Alg.3.8)	R2L (Alg.3.10)	L2R (Alg.5.7)	R2L (Alg.5.8)	R2L (Alg.5.10)	R2L (Alg.5.11)
No. of loops	$L(k)-1$	$L(k)-1$	$L'(k)-1$	$L'(k)$	$L'(k)-1$	$L'(k)$
Correction step	Yes	Yes	-	Yes	Yes	-
No. of Addition	$L(k)$	$L(k)+2m$	$L'(k)-1$	$L'(k)+$ $2m-2$	$L'(k)+$ $2m-2$	$L(k)'$
No. of m-times	$L(k)-1$	$L(k)-1$ one $(m-2)$-times	$L'(k)-1$	$L'(k)$	$L'(k)-1$	$L'(k)$

5.4.2 High Radix Representation

A complexity comparison of algorithms using high radix is shown in Table 5.9. Recall from Alg 3.10, step 9 and 10 can be rewrote as

$A = (k_{h-1}-1)A$

$A = A + R[m]$;

for $i = m-1$ down to 1.

$R[i] = R[i] + R[i+1]$;

$A = A + R[i]$

end for

$A = A + (m-2)R[1]$;

$A = A + P$;

So for the correction steps, there are $2m+1$ addition and one $(m-2)$-times operation. For proposed Alg 5.9, the correction step can also be rewrote. It contains $2m$ additions.

We define $L(k)$ in table 5.8 as $\lceil \log_m k \rceil$. For traditional high radix number system, where $m \in \{0,1,...m-1\}$. The representation range of k digits is $\left[0, m^k-1\right]$. For elevated binary number system, the representation range of k digits is $\left[\frac{(1-m^k)}{1-m}, \frac{m(1-m^k)}{1-m}\right]$. So we can obtain

Table 5.10: Complexity comparison among high radix algorithms

	Joye's algorithms		Proposed Algorithms		
	L2R (Alg.3.8)	R2L (Alg.3.10)	L2R (Alg.5.7)	R2L (Alg.5.8)	R2L (Alg.5.11)
No. of loops	$L(k)-1$	$L(k)-1$	$L'(k)-1$	$L'(k)$	$L'(k)$
Correction step	Yes	Yes	-	Yes	-
No. of Addition	$L(k)$	$L(k)+2m$	$L'(k)-1$	$L'(k)+$ $2m-2$	$L(k)'$
No. of m-times	$L(k)-1$	$L(k)-1$ one $(m-2)$-times	$L'(k)-1$	$L'(k)$	$L'(k)$

following facts:

$L(k) = L'(k)$, when present value of $\left[0, m^{L(k)}-1\right]$.

$L(k) = L'(k)+1$, when present value of $\left[m^{L(k)}, \frac{m(1-m^{L(k)})}{1-m}\right]$.

We can see from Table 5.9, proposed left-to-right Alg 5.7 has better performance than Alg 3.8, since it does not need a correction step and has one less addition operation.

For right-to-left algorithms, Proposed Alg 5.11 has the best performance in complexity since it has the lest number of addition and number of m-times operation. Plus, it doe s not need a correction step. Proposed Alg 5.10 also has advantage in number of addition compared to Joye's Alg 3.10.

Scalar multiplication using elevated high radix number (left-to-right)

Input: Point $P \in E$, $k = \sum_{i=0}^{h-1} k_i m^i, \quad k_i \in \{1, 2, ...m\}$
Output: $kP \in E$

1: **for** $i = 1$ to m **do**
2: $\quad R[i] = iP$;
3: **end for**
4: $A = k_{h-1}P$;
5: **for** $i = h-2$ down to 0 **do**
6: $\quad A = mA$;
7: $\quad A = A + R[k_i]$;
8: **end for**
9: The final value is $kP = A$.

Scalar multiplication using elevated high radix number (right-to-left) version 1

Input: Point $P \in E$, $k = \sum_{i=0}^{h-1} k_i m^i, \quad k_i \in \{1, 2, ...m\}$
Output: $kP \in E$

1: **for** $i = 1$ to m **do**
2: $\quad R[i] = O_G$;
3: **end for**
4: $A = P$;
5: **for** $i = 0$ to $h-1$ **do**
6: $\quad R[k_i] = R[k_i] + A$;
7: $\quad A = mA$;
8: **end for**
9: $A = \sum_{i=1}^{m} iR[i]$;
10: The final value is $kP = A$.

Scalar multiplication using elevated high radix number (right-to-left) version 1

Input: Point $P \in E$, $k = \sum_{i=0}^{h-1} k_i m^i, \quad k_i \in \{1,2,...m\}$
Output: $kP \in E$
1: **for** $i = 1$ to m **do**
2: $R[i] = O_G$;
3: **end for**
4: $A = P$;
5: **for** $i = 0$ to $h-1$ **do**
6: $R[k_i] = R[k_i] + A$;
7: $A = mA$;
8: **end for**
9: $A = R[m]$;
10: **for** $i = m-1$ down to 1 **do**
11: $R[i] = R[i] + R[i+1]$;
12: $A = A + R[i]$;
13: **end for**
14: The final value is $kP = A$.

Scalar multiplication using elevated high radix number (right-to-left) version 2

Input: Point $P \in E$, $k = \sum_{i=0}^{h-1} k_i m^i, \quad k_i \in \{1,2,...m\}$
Output: $kP \in E$
1: $X = O_G$;
2: $Y = P$;
3: **for** $i = 0$ to $h-1$ **do**
4: $X = X + k_i Y$;
5: $Y = mY$;
6: **end for**
7: The final value is $kP = X$.

For an integer in EBNS of length k (there are k digits in the representation),

$$X = (x_{k-1}x_{k-2}...x_1x_0)_2; \quad \text{where } x_i \in \{1,2\}$$

The maximal representable value is $2^{k+1}-2$ and the minimal representable value can be calculated as $2^k - 1$.

So, the representation range of k digits for this new number system is $\left[2^k - 1, 2^{k+1} - 2\right]$. By contrast, the representation rang of k digits for binary number is $\left[0, 2^k - 1\right]$. So, for the same value, EBNS representation needs one bit less than binary number system.

For example, if we want to represent decimal number 27 in this new number system.

We can write it as $(2211)_2$.

For an integer in elevated high radix number system of length k (there are k digits in the representation),

$$X = (x_{k-1}x_{k-2}\cdots x_1x_0)_m; \text{ where } x_i \in \{1,2,...,m\}$$

The maximal representable value is $\frac{m(1-m^k)}{1-m}$ and the minimal representable value can be calculated as $\frac{(1-m^k)}{1-m}$. So, the representation range of k digits for this new number system is $\left[\frac{(1-m^k)}{1-m}, \frac{m(1-m^k)}{1-m}\right]$. By contrast, the representation rang of k digits for binary number is $\left[0, m^k-1\right]$.

For example, if we want to represent decimal number 27 in this new number system with $m = 3$. We can write it as $(223)_3$.

Joye's left-to-right algorithm (Binary case)

Input: Point $P \in E$, $k = (k_{h-1}k_{h-2}...k_1k_0)_2$, $k_i \in \{0,1\}$, $k_{h-1} = 1$
Output: $kP \in E$

1: $x = (k_{h-2}+1)P$;
2: $y = 2P$;
3: **for** $i = h-3$ down to 0 **do**
4: **if** $k_i = 0$ **then**
5: $x = 2x$; $x = x+P$;
6: **else**
7: $x = 2x$; $x = x+y$;
8: **end if**
9: **end for**
10: $x = x+P$;
11: The final value is $x = kP$.

Joye's right-to-left algorithm (Binary case)

Input: Point $P \in E$, $k = (k_{h-1}k_{h-2}...k_1k_0)_2$, $\quad k_i \in \{0,1\}$, $\quad k_{h-1} = 1$
Output: $kP \in E$

1: $x = k_0P$;
2: $y = P$;
3: $z = P$;
4: **for** $i = 1$ to $h-2$ **do**
5: **if** $k_i = 0$ **then**
6: $z = 2z$; $\quad x = x+z$;
7: **else**
8: $z = 2z$; $\quad y = y+z$;
9: **end if**
10: **end for**
11: $z = x+2y$;
12: The final value is $z = kP$.

Input: Point $P \in E$, $k = (11001)_2$
Output: $x = 25P \in E$

1: $x = P$;
2: $k_3 = 1$; $x = 2x$; $x = x+P$; (Two operations)
3: $k_2 = 0$; $x = 2x$; (One operation)
4: $k_1 = 0$; $x = 2x$; (One operation)
5: $k_0 = 1$; $x = 2x$; $x = x+P$; (Two operations)
6: The final value of x is $x = 25P$.

6 Conclusions and Possible Future Works

6.1 Conclusions

In the thesis, a new binary number system, called Elevated binary number system (EBNS), is proposed. Conversions between this new number system and the conventional binary number is discussed. An extension of EBNS to high radix system is also presented.

Proposed Alg 5.1 is the new left-to-right scalar multiplication algorithm using EBNS. Compared to the best existing work, it shows advantage in terms of computation efficiency while maintains the same security strength against SCAs.

We propose five right-to-left algorithms. All of them have an advantage of not having a constraint compared to the best existing work. Also, Alg 5.3 has the lowest number of operations among all algorithms and it offers more security strength since it can resist to Park's fault attack while all existing work can not.

Proposed Alg 5.7 is the left-to-right scalar multiplication algorithm using elevated high radix. Compared to the best existing work described in [7], the main superiority of this algorithm is to reduce complexity while not sacrificing the security strength. We also propose three right-to-left algorithms. Among them, Alg 5.11 can resist to Park's fault attack while all existing works cannot and it also has a better performance in complexity since it has less number of operations and does not have a constraint.

6.2 Possible Future Works

In the future, the research works presented in this thesis can be further extended in the following aspects:

- Hardware implementation of proposed algorithms are expected to be accomplished.
- Algorithms that convert between elevated high radix number system and traditional high radix number system will be developed. Along with their architectures.

- All proposed right-to-left scalar multiplication algorithms have a correction step at the end. This unique feature of algorithms may become a target to certain SCAs. The next goal of our work is to restructure the algorithms in order to eliminate the correction step.

REFERENCES

[1] L. M. Adleman, P. W. Rothemund, S. Roweis, and E. Winfree, "On applying molecular computation to the data encryption standard," *Journal of Computational Biology*, vol. 6, no. 1, pp. 53–63, 1999.

[2] F.-X. Standaert, "Introduction to side-channel attacks," in *Secure Integrated Circuits and Systems*. Springer, 2010, pp. 27–42.

[3] P. Rohatgi, "Electromagnetic attacks and countermeasures," in *Cryptographic Engineering*. Springer, 2009, pp. 407–430.

[4] S.-M. Yen, L.-C. Ko, S. Moon, and J. Ha, "Relative doubling attack against montgomery ladder," in *Information Security and Cryptology-ICISC 2005*. Springer, 2006, pp. 117–128.

[5] 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.

[6] J. Park, K. Bae, S. Moon, D. Choi, Y. Kang, and J. Ha, "A new fault cryptanalysis on montgomery ladder exponentiation algorithm," in *Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human*. ACM, 2009, pp. 896–899.

[7] M. Joye, "Highly regular m-ary powering ladders," in *Selected Areas in Cryptography*. Springer, 2009, pp. 350–363.

[8] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[9] V. S. Miller, "Use of elliptic curves in cryptography," in *Advances in Cryptology-CRYPTO85 Proceedings*. Springer, 1985, pp. 417–426.

[10] P. C. Kocher, "Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems," in *Advances in CryptologyCRYPTO96*. Springer, 1996, pp. 104–113.

[11] D. Brumley and D. Boneh, "Remote timing attacks are practical," *Computer Networks*, vol. 48, no. 5, pp. 701–716, 2005.

[12] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Annual International Cryptology Conference*. Springer, 1999, pp. 388–397.

[13] J. Cathalo, F. Koeune, and J.-J. Quisquater, "A new type of timing attack: Application to gps," in *Cryptographic Hardware and Embedded Systems-CHES 2003*. Springer, 2003, pp. 291–303.

[14] S.-M. Yen and M. Joye, "Checking before output may not be enough against fault-based cryptanalysis," *Computers, IEEE Transactions on*, vol. 49, no. 9, pp. 967–970, 2000.

[15] Y. Sung-Ming, S. Kim, S. Lim, and S. Moon, "A countermeasure against one physical cryptanalysis may benefit another attack," in *Information Security and CryptologyI-CISC 2001*. Springer, 2001, pp. 414–427.

[16] J.-M. Schmidt and C. Herbst, "A practical fault attack on square and multiply," in *Fault Diagnosis and Tolerance in Cryptography, 2008. FDTC'08. 5th Workshop on*. IEEE, 2008, pp. 53–58.

[17] C. H. Kim and J.-J. Quisquater, "Fault attacks for crt based rsa: New attacks, new results, and new countermeasures," in *IFIP International Workshop on Information Security Theory and Practices*. Springer, 2007, pp. 215–228.

[18] W. Van Eck, "Electromagnetic radiation from video display units: An eavesdropping risk?" *Computers & Security*, vol. 4, no. 4, pp. 269–286, 1985.

[19] D. Genkin, A. Shamir, and E. Tromer, "Acoustic cryptanalysis," *Journal of Cryptology*, pp. 1–52, 2016.

[20] A. Shamir and E. Tromer, "Acoustic cryptanalysis," *presentation available from http://www. wisdom. weizmann. ac. il/ tromer*, 2004.

[21] J.-S. Coron, "Resistance against differential power analysis for elliptic curve cryptosystems," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 1999, pp. 292–302.

[22] P.-A. Fouque and F. Valette, "The doubling attack–why upwards is better than downwards," in *Cryptographic Hardware and Embedded Systems-CHES 2003*. Springer, 2003, pp. 269–280.

[23] P. L. Montgomery, "Speeding the pollard and elliptic curve methods of factorization," *Mathematics of computation*, vol. 48, no. 177, pp. 243–264, 1987.

[24] S.-M. Yen, L.-C. Ko, S. Moon, and J. Ha, "Relative doubling attack against montgomery ladder," in *Information Security and Cryptology-ICISC 2005*. Springer, 2005, pp. 117–128.

[25] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi, "Towards sound approaches to counteract power-analysis attacks," in *Advances in CryptologyCRYPTO99*. Springer, 1999, pp. 398–412.

[26] G. Fumaroli and D. Vigilant, "Blinded fault resistant exponentiation," in *Fault Diagnosis and Tolerance in Cryptography*. Springer, 2006, pp. 62–70.

[27] F. Muller and F. Valette, "High-order attacks against the exponent splitting protection," in *Public Key Cryptography-PKC 2006*. Springer, 2006, pp. 315–329.

[28] C. Herbst and M. Medwed, "Using templates to attack masked montgomery ladder implementations of modular exponentiation," in *Information Security Applications*. Springer, 2008, pp. 1–13.

[29] Y. He, "Highly secure cryptographic computations against side-channel attacks," 2013.

VITA AUCTORIS

NAME:	Yue Huang
PLACE OF BIRTH:	Liaoning, China
YEAR OF BIRTH:	1990
EDUCATION:	North China Electric Power University, Beijing, China Bachelor of Electrical Engineering and Management 2008-2012
	University of Windsor, Windsor, ON, Canada Master of Applied Science, Electrical and Computer Engineering 2014-2017