Cryptographic Security: Countermeasures against Side-Channel Attacks

BY

KUN MA
B.S. Jilin University, Changchun, China, 2006
M.S. Beijing University of Posts and Telecommunications, Beijing, China 2009

THESIS

Submitted as partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Chicago, 2014

Defense Committee:

Wenjing Rao, Chair and Advisor
Zhichun Zhu
Ashfaq Khokhar
Venkatakrishnan Venkatesan Natarajan, Computer Science
Kaijie Wu, Chongqing University

ACKNOWLEDGMENTS

I would like to thank my advisors, Professor Wenjing Rao and Professor Kaijie Wu, for all their guidance, support and encouragement throughout my Ph.D study and research. Professor Wu introduced me to the interesting field of side channel attacks and countermeasures. Discussing with him has always been inspiring and helpful. My research work would not be made possible without the guidance of Professor Wu. We have been keeping research interations after he left University of Illinois at Chicago. I'm very grateful for his thoughtful support during the tough times. Professor Rao has been my advisor after Professor Wu left. Her insightful comments and suggestions on my research are important to me towards better work. I'm really grateful for her support in my last year of Ph.D study.

My sincere thanks also goes to my dissertation committee, Professor Zhichun Zhu, Professor Ashfaq Khokhar, Professor Venkat Venkatakrishnan, for valuable comments and suggestions. Special thanks to Professor Zhichun Zhu, who overcame every difficulties to join my dissertation defense even when she was on sabbatical and Professor Ashfaq Khokhar for his help in offering financial support when he was the Director of Graduate Studies in University of Illinois at Chicago.

Thanks to my family. My husband and my parents have been unconditionally supporting and encouraging me. They have always been the source of my strength.

ACKNOWLEDGMENTS (Continued)

KM

TABLE OF CONTENTS

CHAPTER			PAGE
1	**INTRODUCTION**		1
	1.1	Cryptography	1
	1.2	Public Key Cryptography and Applications	5
	1.3	Security of Public Key Cryptography: A Mathematical Point of View	11
	1.4	Side Channel Attacks	12
	1.4.1	Timing Attacks	13
	1.4.2	Power Analysis Attacks	14
	1.4.3	Electromagnetic Analysis Attacks	15
	1.4.4	Fault Attacks	15
	1.5	Contributions	16
2	**PRELIMINARIES**		19
	2.1	Finite Fields	19
	2.1.1	Prime Field	20
	2.1.2	Binary Finite Field	20
	2.2	Elliptic Curves	23
	2.3	Elliptic Curve Scalar Multiplication	26
	2.3.1	Basic Algorithms	26
	2.3.2	Montgomery Ladder Algorithm	26
	2.3.3	Fast ECSM over $\mathbb{F}_{2^m}$	28
	2.4	Modular Exponentiation	30
	2.4.1	Algorithms	30
	2.4.2	Computing RSA Fast with Chinese Remainder Theorem	32
3	**SIDE CHANNEL ATTACKS AND COUNTERMEASURES FOR PUBLIC KEY CRYPTOSYSTEMS**		34
	3.1	Fault Attacks and Countermeasures	34
	3.1.1	Fault Injection and Fault Models	35
	3.1.2	Fault Attacks on ECC and RSA	36
	3.1.3	Countermeasures against Fault Attacks	40
	3.2	Power Analysis Attacks and Countermeasures	42
	3.2.1	Power Analysis Attacks on ECC and RSA	42
	3.2.2	Countermeasures against Power Analysis Attacks	45
4	**A NEW COUNTERMEASURE AGAINST FAULT ATTACKS FOR ECC**		48

TABLE OF CONTENTS (Continued)

CHAPTER **PAGE**

4.1 The Proposed Low-cost Error Detection and Recovery Scheme 48
4.1.1 The Idea 49
4.1.2 The LOEDAR Scheme 52
4.2 Analysis of Error Detection Capability 55
4.2.1 Single Error Occurrence in ECSM/EDR 58
4.2.2 Multiple Error Occurrences in ECSM/EDR 66
4.3 Resistance to Fault Attacks 70
4.4 Extendibility to Thwart Power Analysis Attacks 70
4.5 The Experiments 74
4.5.1 Overhead 74
4.5.2 Comparison to Existing Schemes 77
4.5.3 Error Detection and Recovery Test with Random Fault Injection 79

5 A NEW COUNTERMEASURE AGAINST FAULT ATTACKS FOR RSA 81
5.1 Overview 81
5.2 The Proposed Concurrent Error Detection Scheme 83
5.2.1 The Basic CED Scheme 84
5.2.2 The Modified CED Scheme 86
5.2.3 The Enhanced CED Scheme 88
5.2.4 One for All: Unifying the CED Schemes 89
5.3 Error Detection Capability and Resistance to Fault Attacks . 90
5.3.1 Intentional Faults 91
5.3.2 Accidental Faults 93
5.4 Performance and Cost 93
5.4.1 Performance 95
5.4.2 Cost 98

6 TOWARDS A COMPREHENSIVE COUNTERMEASURE AGAINST MULTIPLE SIDE CHANNEL ATTACKS 101
6.1 Overview 101
6.2 Basic Principles to Construct a Flexible Comprehensive Scheme 103
6.3 A Comprehensive Protection Scheme for RSA 104
6.4 Techniques to Improve Resistance and Performance 107
6.4.1 Register Transfer Level Techniques 108
6.5 Mask Reusing 117
6.6 Security Analysis 120
6.6.1 Resistance to Fault Attacks 120
6.6.2 Resistance to Power Analysis Attacks 126
6.7 Experiment Results 129
6.7.1 Hardware Cost and Performance 130
6.7.2 Simulation of Power Analysis Attacks 130

TABLE OF CONTENTS (Continued)

CHAPTER		PAGE
7	**CONCLUSION**	136
	CITED LITERATURE	139
	VITA	148

LIST OF TABLES

TABLE		PAGE
I	KEY LENGTH VS. SECURITY LEVEL	4
II	SECURITY STRENGTH TIME FRAMES	5
III	THE $F_A(O)$ AND $F_B(O)$ OF THE POINT ADDITION OF SCALAR MULTIPLICATION	61
IV	THE $Pr_{undetected\ error}$ OF THE POINT ADDITION OF SCALAR MULTIPLICATION	64
V	LOEDAR'S COMPATIBILITY WITH COUNTERMEASURES AGAINST POWER ANALYSIS	72
VI	COMPARISON OF COUNTERMEASURES WHEN APPLIED TO LOEDAR	74
VII	THE AREA AND DELAY OF THE ARITHMETIC UNITS	75
VIII	HARDWARE OVERHEAD	75
IX	TIME OVERHEAD AND SWITCHING ACTIVITY OVERHEAD	76
X	COMPARISON OF THE AREAS OF THE ARITHMETIC UNITS	78
XI	COMPARISON OF THE AREAS OF THE COMPLETE SCHEMES	78
XII	COMPARISON OF PERFORMANCE	79
XIII	MAJOR STEPS OF CRT-RSA, VILIGANT ALGORITHM AND GIGRAUD ALGORITHM	94
XIV	PERFORMANCE COMPARISON OF CRT-RSA, VIGILANT ALGORITHM AND GIRAUD ALGORITHM AND OUR CED SCHEME	96
XV	COMPARISON OF THE HARDWARE OVERHEAD OF MODULUS EXTENSION	98

LIST OF TABLES (Continued)

TABLE		PAGE
XVI	COMPARISON OF THE REQUIRED MEMORY	99
XVII	COMPUTATIONAL OVERHEADS OF DIFFERENT MASK UPDATING FREQUENCIES	120
XVIII	RESISTANCE TO POWER ANALYSIS ATTACKS	128
XIX	THE AREA AND PERFORMANCE OF THE PROPOSED COMPREHENSIVE SCHEME	130
XX	COMPARISON OF THE WAVEFORM	135

LIST OF FIGURES

FIGURE		PAGE
1	Point updating graph of the Montgomery ladder algorithm	49
2	Point updating graph of the LOEDAR scheme	51
3	The architecture of the LOEDAR scheme	55
4	LOEDAR's protection in the computation flow	56
5	DFG of point addition and point doubling in ECSM and P_v accumulation	59
6	The unified CED scheme	90
7	The nested structure of single-purpose countermeasures in a comprehensive scheme	104
8	The nested structure of the comprehensive scheme	106
9	The systolic architecture of modular multiplier	111
10	An example: the working state of cells in systolic array	112
11	The schedule of cells for interleaving modular multiplications	114
12	The architecture of the modular multiplier	115
13	Implementation of the interleaving modular multiplier	116
14	The circuit of a cell the modular multiplier	116
15	The hardware architecture of the comprehensive scheme	117
16	Mask reusing for consecutive messages	118
17	Similation of the power trace using toggle counts	132
18	Power trace of middle iterations for computing modular exponentiation	133

LIST OF FIGURES (Continued)

FIGURE **PAGE**

19 Power consumption of exponentiating m and m^2 134

SUMMARY

Side channel attacks have become a serious threat to a cryptosystem. Even if a cryptographic algorithm is mathematically secure, its software or hardware implementation may leak information through side channels. Side channel attacks exploit the side channel leakage to break a cryptosystem. A small amount of side channel information may be sufficient to compromise a cryptosystem which cannot be broken by pure cryptanalysis with current computing capability in limited time. As side channel attacks are not only powerful but also practical, protecting cryptosystem against side channel attacks is critical to ensure security in cryptographic applications.

This work aims at protecting public key cryptosystems against side channel attacks. Fault attack is one of the most powerful side channel attacks. In fault attacks, an attacker needs to induce faults to disturb the cryptosystem and reveal the secret information by analyzing the erroneous output or the reaction of the cryptosystem under faults. We present countermeasures against fault attacks for two widely used public key cryptosystems, Elliptic Curve Cryptography (ECC) and RSA. The countermeasures prevent the attacker from obtaining erroneous result through error detection. Although parallel computation with two identical datapaths or recomputation can verify the correctness of result, the high overhead in terms of hardware or time may not be acceptable. In this dissertation, we propose low-cost error detection methods for ECC and RSA as countermeasures against fault attacks.

The error detection method for ECC shares the same design principle as the error detection method for RSA. We discover or construct an invariant in the ECC or RSA algorithm based on mathematical properties. The invariant is verified to detect errors. In the proposed error detection scheme for ECC, an invariant is constructed by introducing a new point in the Montgomery ladder algorithm. Meanwhile, it retains the property to compute ECC fast. The proposed scheme also supports fast recovery upon detecting errors. It saves the intermediate results as a checked state after their correctness is verified. Upon detecting errors, it can easily roll back to the previous checked state and restart computation from there rather than from the very beginning. The process reduces the loss in time to recover from error and hence improves the overall efficiency. This is desirable in mission-critical systems which are required to deliver service on time even in the events of natural or attacker-induced faults. The proposed error detection scheme for RSA exploits the multiplicative homomorphic property of RSA to construct an invariant. The invariant is verified for every k messages to detect errors. k can be adjusted to trade-off time overhead with storage overhead and output latency according to requirement of applications.

When designing the error detection schemes against fault attacks for ECC and RSA, we also take into account their extendibility to thwart other types of side channel attacks, e.g., power analysis attacks. In practice, public key cryptosystems are threatened by all types of side channel attacks since an attacker is able to choose any attacking approach. Protecting a cryptosystem against fault attacks is not sufficient to ensure security. Hence, extendibility or compatibility is important for a countermeasure to be useful in the face of various side channel

attacks. The proposed scheme for ECC is compatible with most of the existing countermeasures against power analysis attacks. It is easy to extend it to thwart power analysis attacks. The proposed error detection scheme for RSA does not impose any requirement on how to compute modular exponentiation. Hence, it can work with any RSA architecture which could be a fine-tuned architecture resistant to other types of side channel attacks.

Although various countermeasures have been proposed to thwart side channel attacks, most of them target only one type of side channel attacks. With a rich variety of such single-purpose countermeasures, we discuss how to construct a comprehensive countermeasure using the single-purpose countermeasures to thwart multiple types of side channel attacks. Combining the single-purpose countermeasures carelessly may create new leakage. We propose the basic principles to choose single-purpose countermeasures and construct a comprehensive one without compromising the security of any single-purpose countermeasure. Following the principles, we present a comprehensive scheme for RSA to thwart two of the most powerful side channel attacks, fault attacks and power analysis attacks. Techniques at hardware level are used to enhance the resistance to power analysis attacks and improve the overall performance of the scheme. Resistance to other types of side channel attacks can be integrated flexibly under the proposed principles. The constructing method can be used to create a comprehensive scheme for ECC cryptosystem.

CHAPTER 1

INTRODUCTION

Cryptography has been applied to secure communication for thousands of years. It enables communicating parties to exchange sensitive information securely over an insecure communication channel by encrypting messages. With the development of Internet and electronic techniques, cryptography is not limited to just a set of encryption mechanism. Modern cryptography is developed to provide new security properties for various applications and services. Examples include digital signature to demonstrate the authenticity of a digital document, user authentication to control access to some restricted services, data protection to prevent unauthorized use or reproduction of copyrighted files, etc.

1.1 Cryptography

Modern cryptography can provide confidentiality, data integrity, authentication and non-repudiation. The purpose of confidentiality is to guarantee that the information cannot be disclosed by unauthorized users. Confidentiality is enforced by encryption. When two or more parties have to exchange sensitive information over an insecure communication channel, the communication parties encode sensitive information to a serial of unreadable codes using encryption algorithm, send them over the communication channel. Only the authorized parties can decode it. The received message, after decoded, should be exactly the same one sent by the sending party. This is referred to as data integrity. Accuracy and consistency of data should be

maintained, and unauthorized modification violates data integrity. Besides verification of data integrity, the receiver should be able to verify the origin of data. It is achieved by authentication. Communication parties are able to validate identification each other through authentication. Non-repudiation prevents the sender of information from denying having sent it.

Different security goals can be achieved by different cryptographic schemes such as hash functions, secret key cryptography, public key cryptography, etc. Hash functions can be applied to verify data integrity. Hash function maps an arbitrary finite size of block data to a string of fixed length. Any modification of data will generally lead to different hash values. The Secure Hash Algorithm (SHA) is a family of hash functions published by National Institute of Standards and Technology (NIST) as Federal Information Processing Standards (FIPS). Secret key cryptography is also called symmetric key cryptography in which a single, shared, secret piece of information, called key, is used to encrypt and decrypt data. Data Encryption Standard (DES) and Advanced Encryption Standard (AES) are two well-known secret key cryptographic algorithms. AES has been announced as FIPS to supersede DES. Secret key encryption is fast and hence is suited to encryption of high volume of data. It can be used to encrypt messages in secure communication. When the communicating parties are located far away from each other, they have to agree on a secret key before encrypting messages. The exchange of secret key is addressed by public key cryptography. Public key cryptography, or asymmetric key cryptography, involves a pair of different keys, a public key and a private key. The public key is published openly and the private key is kept secret. Besides key establishment for secret key encryption, public key cryptography also plays an important role in digital signature. Digital

signature addresses data authentication and non-repudiation. Analogous to ink signature in the real world, digital signature is used to authenticate signer in electronic world. A signer generates a signature for an electronic document using the private key, and anyone can verify the signature with the corresponding public key. Ronald Rivest, Adi Shamir and Leonard Adleman presented the first public key cryptographic scheme, RSA, in 1978 (1). RSA has been used as public key cryptographic scheme for decades. Elliptic Curve Cryptography, an alternative public key cryptographic approach, was introduced independently by Neal Koblitz (2) and Victor Miller (3) in 1985. NIST has included the schemes based on ECC in its recommended cryptographic algorithms (4).

In modern cryptography, key is an essential element to provide security properties. It is used in cryptographic algorithms to encrypt the plaintext or to decrypt the ciphertext. In cryptographic terminology, plaintext is the message to be encrypted and ciphertext is the encryption of plaintext. According to the basic principle of modern cryptography, Kerckhoffss principle, the security of a cryptosystem should depend on the secrecy of the key and not on the secrecy of cryptographic algorithm. Most modern cryptographic algorithms are designed under the assumption that the adversaries will gain full knowledge of the cryptosystem. In other words, the design of cryptosystem is not required to be secret. As a result, modern cryptography provides secrecy by just keeping relatively small keys secret. Key length is an important security parameter which is measured in bits of the key. It is always used to indicate the security strength of a cryptosystem, such as 256-bit AES, 2048-bit RSA. Generally speaking, increasing key length will gain the security strength, for example, 2048-bit RSA provides higher

TABLE I

KEY LENGTH VS. SECURITY LEVEL

Security strength (Bits)	ECC key length (Bits)	RSA key length (Bits)	AES key length (Bits)
80	163-223	1024	-
112	224-255	2048	-
128	256-383	3072	128
192	384-511	7680	192
256	512+	15360	256

security than 1024-bit RSA. However, secret key cryptography and public key cryptography cannot be compared directly by key length for security level as they may have different levels of cryptographic complexity. The security levels are only estimated based on specific public key cryptographic algorithms. NIST guidelines give key length verses security level (5), as shown in Table I. 2048-bit RSA provides equivalent strength to 112-bit symmetric key algorithms. RSA requires much longer keys than ECC for the same security level. For example, 224-bit ECC would have the same strength as 2048-bit RSA.

As the length of key determines security level, it should be chosen according to the requirement of cryptographic applications. For security purpose, NIST also recommends the length of keys to ensure security in next two decades (5). Table II shows the recommendations when cryptographic protection is applied to data, where “Acceptable” means that the algorithm or key length is now known to be secure and “Disallowed” indicates that the algorithm or key length shall not be used.

TABLE II

SECURITY STRENGTH TIME FRAMES

Security Strength (Bits)	2014 through 2030	2031 and beyond
80	Disallowed	Disallowed
112	Acceptable	Disallowed
128	Acceptable	Acceptable
192	Acceptable	Acceptable
256	Acceptable	Acceptable

Even though the cryptographic schemes are designed for high theoretical strength, their implementations in hardware or software may leak information. Side channel attacks exploit the information leaked by physical implementation such as timing, power consumption, electromagnetic radiation and erroneous result to break a cryptosystem. In this dissertation, we focus on protecting public key cryptographic schemes against side channel attacks.

1.2 Public Key Cryptography and Applications

Public key cryptography involves a pair of different keys, a public key and a private key. The public key is used for encrypting a message. The private key must be known to decrypt a ciphertext. The roles of keys are reversed in digital signature. The private key is used to sign a message, and the signature can be verified with the public key. Without knowing the private key, others should not be able to generate a valid signature.

In 1976, Whitfield Diffie and Martin Hellman presented the public key cryptosystem and discussed one-way functions to develop public key cryptosystem (6). A one-way function is

a function such that it is easy to compute in one direction but very difficult to compute in the other. More precisely, a function f is a one-way function if it is easy to compute the $y = f(x)$ for any argument x, yet, it is computationally infeasible to solve the equation for x given y, i.e., $x = f^{-1}(y)$. If the inverse direction is easy to compute given a certain piece of information (the trapdoor), the one-way function is a trapdoor one-way function. Public key cryptosystems are based on trapdoor one-way functions. The private key gives information about the trapdoor. Lacking the trapdoor, one can only compute the function in the forward direction. The forward direction corresponds to encryption or signature verification. Knowing the trapdoor, one can compute the function in inverse direction which corresponds to decryption or signature generation.

RSA, presented by Rivest, Shamir and Adleman in 1978 (1), is still one of the most popular public key cryptosystems. The encryption function is based on modular exponentiation. A message m is encrypted by exponentiating m with a public key e, i.e., $c = m^e \bmod N$, where N is the product of two large primes p and q. The ciphertext c can be decrypted by computing $m = c^d \bmod N$ with the private key d. In digital signature, the signer performs modular exponentiation $s = m^d \bmod N$ with the private key d to generate the signature s, and the signature receiver uses the signer's public key e to verify the signature by computing $m' = s^e \bmod N$. The signature is correct if and only if $m' \equiv m \bmod N$. The message m may be pre-processed with a padding scheme agreed on by two communicating parties.

To generate the keys for RSA, two distinct prime numbers p and q are randomly selected. The length of p and q should be similar. Modulus N is computed as $N = p \cdot q$, and its Eulers

totient function is $\varphi(N) = (p-1)(q-1)$. Then an integer e is chosen as the public key such that $1 < e < \varphi(N)$, e and $\varphi(N)$ are coprime, i.e., $\gcd(e, \varphi(N)) = 1$. The private key d is computed as $d = e^{-1} \bmod \varphi(N)$. Since p, q and $\varphi(N)$ are used to compute the private key, they must also be kept secret besides the private key itself. For security reasons, p and q should be chosen with additional requirements described in Public Key Cryptography Standards (PKCS) #1 (7).

The correctness of RSA can be proved with Fermats little theorem. The theorem states that if p is a prime number and a is not divisible by p, then $a^{p-1} \equiv 1 \bmod p$. According to the key generation procedure of RSA, the public key and the private key satisfy $ed \equiv 1 \bmod \varphi(N)$ where $\varphi(N) = (p-1)(q-1)$. By the definition of modular operation, $ed - 1 = i(p-1)(q-1)$ for some nonnegative integer i. In RSA signature scheme, the signature is verified by computing $s^e \bmod N = (m^d)^e \bmod N = m^{ed} \bmod N$. In RSA encryption scheme, the ciphertext is decrypted by computing $c^d \bmod N = (m^e)^d \bmod N = m^{ed} \bmod N$. Using Fermats little theorem, we have

$$m^{ed} = m \cdot m^{(ed-1)} = m \cdot m^{i(p-1)(q-1)} = m \cdot (m^{(p-1)})^{i(q-1)} \equiv m \bmod p \tag{1.1}$$

Similarly, we have

$$m^e d = m \cdot m^{(ed-1)} = m \cdot m^{i(p-1)(q-1)} = m \cdot (m^{(q-1)})^{i(p-1)} \equiv m \bmod q \tag{1.2}$$

Therefore, $m^e d \equiv m \bmod p \cdot q$.

As an alternative to RSA, ECC can achieve strong security level with relatively small keys. According to Table I, one should use 3072-bit RSA for 128-bit AES, while the equivalent key length for ECC is only 256 bits. ECC offers more security per bit increase in key length than RSA. Although elliptic curve arithmetic for ECC is slightly more complex than RSA, ECC is more computational efficient than RSA due to the increased security strength per bit of ECC.

ECC performs operations on elliptic curve. An elliptic curve is a set of points satisfying a curve equation along with a point at infinity. For cryptographic purpose, the elliptic curve is defined over a finite field. Elliptic curve arithmetic will be discussed in §2.2. Given a set of parameters which defines an elliptic curve and a base point P on the elliptic curve, the key pair (d, Q) is generated by a randomly selected non-zero integer d and a multiplication $Q = dP$. d is the private key and required to be kept secret, and Q is the public key which can be made public. The key generation procedure involves Elliptic Curve Scalar Multiplication (ECSM) which multiplies a point P on the elliptic curve with a scalar k, i.e., kP. The result is also a point on the curve. ECSM is the core operation in ECC-based cryptographic implementations.

In the Elliptic Curve Digital Signature Algorithm (ECDSA), which was standardized in FIPS 186-4 (8), the signer generates a key pair (d, Q) where d is the private signing key and $Q = dP$ is the public key for signature verification. The signer chooses a per-message random integer k such that $1 \leq k \leq n - 1$ where n is the order of base point P, i.e., $n = ord(P)$. Then $kP = (x_1, y_1)$ is computed, and the result x_1 is converted to integer $\overline{x_1}$ to compute $r = \overline{x_1} \bmod n$. The message m is hashed to a bit string of length no more than n and is then converted to an integer e. Then s is computed as $s = k^{-1}(e + dr) \bmod n$. The signature for message m is (r, s).

Neither r nor s can be zero, otherwise the signature should be recomputed by choosing a new random integer k. To verify the received signature (r, s), the received message m is hashed and converted to an integer e. Then point X is computed as $X = (x_1, y_1) = u_1P + u_2Q$, where $u_1 = ew \bmod n$, $u_2 = rw \bmod n$ and $w = s^{-1} \bmod n$. The signature is valid if and only if $r = \overline{x_1} \bmod n$ where $\overline{x_1}$ is the integer computed from x_1.

The signature verification procedure can authenticate the signer. Suppose signature (r, s) is generated for message m. According to the signature generation procedure, $s = k^{-1}(e + dr) \bmod n$. Hence $k = s^{-1}(e + dr) \bmod n$. In the verification procedure,

$$u_1P + u_2Q = (u_1 + u_2d)P = s^{-1}(e + dr)P = kP \tag{1.3}$$

Hence, only the signature generated by the message owner who has the private key can pass the verification procedure. It is crucial that the per-message random number k is kept secret, otherwise the private key d can be revealed by computing $d = r^{-1}(ks - e) \bmod n$. The random number k should be used to sign only one message. If it is used to sign two different messages m_1 and m_2 with the same private key d and generate two signatures (r, s_1) and (r, s_2), then k can be computed by $k = (s_1 - s_2)^{-1}(e_1 - e_2) \bmod n$. Hence, d can be recovered by $d = r^{-1}(ks - e) \bmod n$.

The Diffie-Hellman key exchange based on ECC, Elliptic Curve Diffie-Hellman (ECDH), is a variant of Diffie-Hellman key exchange protocol which enables communicating parties to establish a shared secret key for secret key encryption over an insecure communication channel.

Suppose Alice and Bob are two communicating parties who want to exchange secret. They have to agree on a set of elliptic curve and domain parameters and each generates a pair of keys, i.e., a public key and a private key, for operation. Alice may randomly select an integer as her private key d_A and compute $Q_A = d_A P$ as her public key, where P is the base point on the elliptic curve agreed upon. Similarly, Bob generates his key pair (d_B, Q_B). Then Alice and Bob exchange their public keys over the insecure channel. Alice receives Bobs public key Q_B and she computes $(x, y) = d_A Q_B$. Bob receives Alices public key Q_A and he computes $(x, y) = d_B Q_A$. The value (x, y) computed by both parties are equal because $d_A Q_B = d_A d_B P = d_B d_A P = d_B Q_A$. Hence, it is a shared secret between Alice and Bob. The shared secret can be directly used as the secret key, but care should be taken to remove weak bits. Most standardized ECDH-based protocols derive the secret key from the shared secret.

ECDH immediately leads to the classic Elliptic Curve El-Gamal (EC-ElGamal) encryption. To encrypt a message m, a random number r is chosen, $1 < r < n$ where $n = ord(P)$. Then we compute $R = rP$ and $c = rQ + m$ with the public key Q. The ciphertext is (R, c). To decrypt the ciphertext, we compute $m = c - dR$ with the private key d $(Q = dP)$. The classic EC-ElGamal encryption scheme is insecure against active attackers (9). Moreover, it requires that a mesage should be transformed to a point on the elliptic curve. This problem can be addressed by hashed ElGamal or the Elliptic Curve Integrated Encryption Scheme (ECIES). The hashed ElGamal appears in the ANSI X9.63 standard (10). In contrast to the classic EC-ElGamal, it applies a key derivation function H to rQ, i.e., $k = H(rQ)$ where r is a random number and Q is the public key $(Q = dP)$. And c is computed with a secret key encryption

scheme, $c = \mathcal{E}_k(m)$. The ciphertext is (R, c) where $R = rP$. To decrypt the ciphertext, we compute $k = H(d \cdot R)$ and $m = \mathcal{D}_k(c)$.

ECIES is the same as hashed ElGamal except that it includes Message Authentication Code (MAC) to prevent active attacks. It also appears in ANSI X9.63 standard (10).

1.3 Security of Public Key Cryptography: A Mathematical Point of View

Public key cryptosystems are based on one-way functions which are believed to be computational infeasible to invert. RSA computes exponentiation modulo a composite number N. The composite number N is the product of two large primes p and q. Given p and q, it is easy to compute $N = p \cdot q$. However, inverting this function is difficult. Inverting the function is to find the factors p and q given N. The factorization problem is a hard mathematical problem when the numbers are very large and p, q are of similar size. No efficient factorization algorithm is known. The difficulty of solving the prime factorization problem is the basis of RSA security. The security of ECC is based on the difficulty of solving the Elliptic Curve Discrete Logarithm Problem (ECDLP). More specifically, given points P and Q on elliptic curve, it finds a number k that $kP = Q$ and k is thus called the discrete logarithm of Q to the base P. As of now, RSA-768 has been successfully factored. The factorization work took almost 2000 2.2GHz-Opteron-CPU years (11). The record of solving ECDLP over 112-bit prime field was announced in 2010. The calculation took around 3.5 months on a cluster of more than 200 PlayStation 3 game consoles (12). The previous record solving the problem over 109-bit prime field was set in 2002 requiring 549 days of computing by more than 10,000 members in almost 250 teams, that is around 4000 to 5000 PC years. However, the key lengths used in practice are

usually at least 1024 bits for RSA and at least 163 bits for ECC. NIST recommends the length of keys for public key cryptosystems to ensure security in next two decades (5). According to Table I and Table II, RSA 2048-bit and ECC at least 224-bit are considered to be secure before year 2030. With the length of key properly chosen based on the NIST recommendation, public key cryptosystem are now considered mathematically secure.

1.4 Side Channel Attacks

In practice, however, public key cryptosystems are not secure enough as they are designed to be. Attackers can exploit the weakness in the software or hardware implementations of public key cryptosystems. The implementations leak additional information via side channels. Typical side channels include timing, power consumption, electromagnetic radiation, erroneous output, etc. A small amount of side channel information may be sufficient to break the cryptosystems. Side channel attacks exploiting side channel information require much less computing resources than mathematical cryptanalysis. They are powerful to break public key cryptosystems which are believed mathematically secure and hence become a serious threat. Side channel attacks can be classified in the following ways.

- *Active vs. passive*: Active attacks attempt to tamper the functionality of cryptographic device. For example, in fault attacks faults are induced to cryptographic device to generate erroneous output. Passive attacks collect side channel information by measuring the physical parameters, observing the behavior of cryptographic device. The functionality of cryptographic device is not disturbed.

- *Invasive vs. non-invasive*: Invasive attacks obtain side channel information in an invasive way such as depackaging the device. Different components of the device can then be accessed directly. In non-invasive attacks, the cryptographic device is intact and external information such as timing, power consumption are exploited.

Based on the types of side channel information exploited by attacks, side channel attacks can also be divided into timing attack, power analysis attack, fault attack, etc.

1.4.1 Timing Attacks

Timing attacks exploit the non-constant execution time of cryptographic algorithm to recover the secret information. The time variation can be caused by conditional branches in the algorithm, performance optimizations, etc. The execution time may be slightly different depending on the input data and secret key. Hence, the secret information can be derived by analyzing the time for each execution.

The timing characteristics in public key cryptosystems are more dependent on key than secret key cryptosystems. Hence, public key cryptosystems are more vulnerable to timing attacks.

The most obvious way to prevent timing attacks is to implement cryptographic algorithms such that the execution time is constant. This is often difficult. Another approach is to make timing measurements inaccurate by randomization techniques such as adding random delays. Attackers have to collect more measurements to compensate the noise caused by random delays. The fast increase of measurements required can make timing attacks infeasible.

1.4.2 Power Analysis Attacks

Nowadays, CMOS technology is the predominant technology for digital integrated circuits. The dynamic power consumption of CMOS circuits depends on the change of data. Switching from logic 0 to logic 1 (i.e., $0 \rightarrow 1$) or from logic 1 to logic 0 (i.e., $1 \rightarrow 0$) consumes dynamic power, while staying at the same logic level (i.e., $0 \rightarrow 0$ and $1 \rightarrow 1$) does not. Power analysis attacks exploit the dependence of the dynamic power consumption of a cryptographic device on the data it is processing to recover secret information.

Simple Power Analysis (SPA) attacks require only one power consumption trace over time. In SPA, an attacker identifies the key-dependent operations by observing the power consumption trace. Differential Power Analysis (DPA) attacks involve statistical analysis of correlation between the power consumption and the processed data. In DPA, an attacker makes a hypothesis on the partial key and computes the intermediate values from the key. The intermediate values are then transformed to leakage through a leakage model, e.g., Hamming weight model or Hamming distance model. The attacker performs statistical tests, e.g., distance-of-means, correlation analysis, on the leakage and the power consumption with a large number of power traces. A large correlation should appear for the correct hypothesis.

To prevent SPA attacks, key-dependent operations should be avoided or made indistinguishable in power consumption. DPA attacks can be prevented by hiding the power consumption or masking the data. Hiding reduces the correlation between intermediate data and power consumption by adding noise, reducing signal leakage or using a variable clock frequency. Masking

makes intermediate data unpredictable, thereby breaking the correlation between intermediate data and power consumption.

More details on power analysis attacks on public key cryptosystems and countermeasures will be discussed in Chapter 3.

1.4.3 Electromagnetic Analysis Attacks

Electromagnetic analysis (EMA) attacks exploit the leakage of electromagnetic fields due to current flows. Quisquater and Samyde presented the first EMA attack (13). They measured the electromagnetic radiation of a smart card with an oscilloscope, a flat coil and a Faraday cage. The measurement can be analyzed in a similar way to power analysis attacks. And similar to power analysis attacks, there are simple electromagnetic analysis (SEMA) attacks and differential electromagnetic analysis (DEMA) attacks.

EM radiation can be measured from a distance. Hence, the measurement may be more noisy compared to the power measurement in power analysis attacks.

Countermeasures against EMA attacks include confining the radiation by metal layers, blurring the radiation with an active emitting grid, canceling the radiation using dual logic, etc.

1.4.4 Fault Attacks

In contrast to timing attacks, power analysis attacks and electromagnetic analysis attacks, fault attacks are active attacks which require an attacker to induce faults into cryptographic device when it is performing computation. The erroneous output is analyzed to reveal the secret information.

To thwart fault attacks, error detection can be performed when the cryptographic device is computing. The result will not be outputted once errors are detected. This prevents an attacker from obtaining erroneous result for analysis. More details on fault attacks on public key cryptosystems and countermeasures will be discussed in Chapter 3.

In this dissertation, we consider two of the most powerful side channel attacks, fault attacks and power analysis attacks, and present countermeasures for public key cryptosystems against them.

1.5 Contributions

In this dissertation, we present new countermeasures for two widely used public key cryptosystems, ECC and RSA, to thwart fault attacks. The countermeasures are based on error detection methods. If an error is detected, the cryptosystem will not output the erroneous result and hence prevent an attacker from obtaining it to perform fault analysis. The proposed low-cost error detection and recovery scheme for ECC introduces a new point P_v into the Montgomery ladder algorithm and updates it in a way such that $P_v + P_2 \equiv 2P_1$, where P_1 and P_2 are two point variables used in the Montgomery ladder algorithm. $P_v + P_2 \equiv 2P_1$ is verified to detect errors. The error detection scheme for RSA exploits the homomorphic property of RSA encryption, i.e., $E(m_1 \cdot m_2) \equiv E(m_1) \cdot E(m_2)$, and checks it to detect errors. Both error detection schemes are design from the same idea: constructing an invariant from the mathematical properties of the public key cryptosystems and checking the correctness of the invariant to detect errors. The proposed error detection methods can achieve good error detection capability while significantly reduce the overhead in terms of time and hardware compared to error

detection methods based on parallel computing and recomputing. The idea of discovering or constructing an invariant for error detection should work for other mathematically structured cryptographic cryptosystems besides ECC and RSA. We hope it can provide some guidance for designing the error detection methods for other cryptosystems.

We also discuss the basic principles to construct a comprehensive scheme to thwart multiple side channel attacks. A comprehensive scheme is critical in the security of public key cryptosystems since an attacker can choose any side channel attack in practice and he/she needs to succeed in only one attack to break the public key cryptosystems. Most existing countermeasures target at preventing a single type of side channel attacks. The countermeasures proposed to thwart more than one type of side channel attacks customize the computations, which makes them inflexible to be extended when new side channel attacks appear. Careless modification can cause new leakage compromising the expected security, as shown by Kim and Quisquater (14). Designing a new scheme from scratch whenever resistance to new side channel attacks has to be included is unrealistic. We present an approach to construct a comprehensive scheme from a rich variety of single-purpose countermeasures against different side channel attacks. The scheme can flexibly integrate the countermeasures against new side channel attacks at any time. We show a scheme for RSA constructed with the proposed approach and propose techniques at the Register Transfer Level to improve the performance and the resistant to power analysis attacks.

The rest of the dissertation is organized as follows. In Chapter 2, we introduce the mathematical background, basic definitions and algorithms for the public key cryptosystems, ECC

and RSA. In Chapter 3, we focus on two of the most powerful side channel attacks, fault attacks and power analysis attacks on ECC and RSA. The related work on countermeasures against them is introduced. We present a low-cost error detection and recovery scheme (LOEDAR) for ECC to thwart fault attacks in Chapter 4. It can be easily extended to thwart power analysis attacks. In Chapter 5, we propose a concurrent error detection scheme for RSA to thwart fault attacks, which can work with any fine-tuned RSA architecture resistant to other side channel attacks. To protect the public key cryptosystems against multiple types of side channel attacks, we discuss the basic principles to construct a comprehensive scheme using single-purpose countermeasures and present one such scheme for RSA in Chapter 6. Chapter 7 concludes the dissertation.

CHAPTER 2

PRELIMINARIES

Public key cryptographic algorithms operate over finite fields. These algebraic structures are the mathematical basis to understand public key cryptographic algorithms. They define the arithmetic operations in public key cryptographic algorithms. This chapter gives a brief overview of the finite fields, and introduces elliptic curves and operations on the elliptic curves. Then the algorithms to compute ECSM for ECC and modular exponentiation for RSA are introduced.

2.1 Finite Fields

A field is a set $\mathbb{F}$ with two operations, usually called addition ($+$) and multiplication ($\cdot$), respectively, for which the following conditions are satisfied:

- *Closure*: If $a, b \in \mathbb{F}$, then $a + b \in \mathbb{F}$ and $a \cdot b \in \mathbb{F}$.
- *Associativity*: $a + (b + c) = (a + b) + c$ and $a \cdot (b \cdot c) = (a \cdot b) \cdot c$ for all $a, b, c \in \mathbb{F}$.
- *Commutativity*: $a + b = b + a$ and $a \cdot b = b \cdot a$ for all $a, b \in \mathbb{F}$.
- *Additive identity* and *multiplicative identity*: There exists an element, denoted by 0, $0 \in \mathbb{F}$ such that for all $a \in \mathbb{F}$, $0 + a = a + 0 = a$. There also exists an element, denoted by 1, $1 \in \mathbb{F}$ such that for all $a \in \mathbb{F}$, $1 \cdot a = a \cdot 1 = a$.
- *Additive inverse* and *multiplicative inverse*: For any $a \in \mathbb{F}$, there exists $-a \in F$ such that $a + (-a) = 0$. For any $a \in \mathbb{F}$ and $a \neq 0$, there exist $b \in \mathbb{F}$ such that $a \cdot b = b \cdot a = 1$.

- *Distributivity*: $a \cdot (b + c) = a \cdot b + a \cdot c$ and $(b + c) \cdot a = b \cdot a + c \cdot a$ for all $a, b, c \in \mathbb{F}$.

According to the above definition of field, the set of all real numbers is a field, denoted by $\mathbb{R}$, and the set of all rational numbers is also a field, denoted by $\mathbb{Q}$. There are infinite elements in $\mathbb{R}$ and $\mathbb{Q}$. A field containing a finite number of elements is called finite field or Galois field. The order of finite field, i.e., the number of elements in the field, is equal to p^m for some prime p and positive integer m. p is called the characteristic of the field. Hence, the finite field can be denoted by $\mathbb{F}_{p^m}$. When $m = 1$, the finite field is called prime field and denoted by $\mathbb{F}_p$. When $p = 2$, the field is known as binary extension finite field or simply binary finite field, denoted by $\mathbb{F}_{2^m}$. Prime field and binary finite field play an important role in public key cryptography.

2.1.1 Prime Field

Given a positive integer n, an integer i can be expressed as $i = qn + r$ for some integer r, $0 \leq r \leq n - 1$, and some integer q. r, namely the remainder, is i modulo n, i.e., $r = i \bmod n$. For prime p, the set of mod-p remainders contains p integers, $R_p = \{0, 1, \cdots, p - 1\}$. The remainder set R_p forms a prime field $\mathbb{F}_p$ under mod-p addition and mod-p multiplication. The mod-p addition and mod-p multiplication are addition and multiplication performed modulo p. The prime field $\mathbb{F}_p$ has a prime p number of elements.

2.1.2 Binary Finite Field

Binary finite field $\mathbb{F}_{2^m}$ can be constructed with polynomials. A polynomial in X over the field $\mathbb{F}$ is an expression of the form

$$a(X) = a_n X^n + a_{n-1} X^{n-1} + \cdots + a_1 X + a_0$$

where $a_n,\ a_{n-1},\ \cdots,\ a_1,\ a_0 \in \mathbb{F}$ and n is a positive integer. $a_n,\ a_{n-1},\ \cdots,\ a_1,\ a_0$ are the coefficients of the polynomial. The largest i for which a_i is nonzero is called the degree of the polynomial. A polynomial $a(X)$ is said to be irreducible if $degree(a(X)) \geq 1$ and $a(X)$ is divisible only by polynomials of degree 0 or $c \cdot a(X)$ for some $c \in \mathbb{F}$.

The set of all polynomials over the field $\mathbb{F}$ in X is denoted by $\mathbb{F}[X]$. The field $\mathbb{F}_{p^m}$, when $m \neq 1$, can be obtained by taking $\mathbb{F}_p[X]$, the set of all polynomials with coefficients over the prime field $\mathbb{F}_p$, modulo an irreducible polynomial of degree m. Hence, the binary finite field $\mathbb{F}_{2^m}$ can be constructed by taking $\mathbb{F}_2[X]$ modulo an irreducible polynomial of degree m. Hence, the elements in $\mathbb{F}_{2^m}$ are the polynomials of degree at most $m-1$ and coefficients of the polynomials are in the field $\mathbb{F}_2 = \{0, 1\}$. For example, the binary finite field $\mathbb{F}_{2^4}$ contains 16 polynomials: 0, 1, X, $X+1$, X^2, X^2+1, X^2+X, X^2+X+1, X^3, X^3+1, X^3+X, X^3+X+1, X^3+X^2, X^3+X^2+1, X^3+X^2+X, X^3+X^2+X+1. Arithmetic operations over $\mathbb{F}_{2^4}$ are performed using an irreducible polynomial as the reduction polynomial, e.g., $f(X) = X^4+X+1$. The following are examples of addition and multiplication. Suppose $g(X) = X^3+X+1$ and $h(X) = X^2+1$.

$$\begin{aligned} Addition: g(X) + h(X) &= (X^3 + X + 1) + (X^2 + 1) \\ &= X^3 + X^2 + X \end{aligned}$$

$$\begin{aligned} Multiplication: g(X) \cdot h(X) &= (X^3 + X + 1) \cdot (X^2 + 1) \\ &= X^5 + X^2 + X + 1 \bmod (X^4 + X + 1) \\ &= 1 \end{aligned}$$

A binary finite field $\mathbb{F}_{2^m}$ can also be viewed as a vector space of dimension m over $\mathbb{F}_2$. And the elements in the field can be represented in bases such as polynomial bases and normal bases.

Let α be an element of $\mathbb{F}_{2^m}$ such that α is a root of an irreducible polynomial $f(X)$ of degree m over $\mathbb{F}_2$, the element $\alpha \in \mathbb{F}_{2^m}$ generates

$$\{1, \alpha, \alpha^2, \cdots, \alpha^{m-1}\}$$

which is called a polynomial basis of $\mathbb{F}_{2^m}$ over $\mathbb{F}_2$. Any element in the field can be represented in terms of the basis as $a(X) = a_{m-1}\alpha^{m-1} + \cdots + a_1\alpha + a_0$. In computer systems, the element can be represented by the vector formed by the coefficients. For example, $X^3 + X^2 + 1$ in the field $\mathbb{F}_{2^4}$ is represented as $(1101)_2$.

A normal basis of $\mathbb{F}_{2^m}$ over $\mathbb{F}_2$ is a basis of the form

$$\{\alpha, \alpha^2, \alpha^{2^2}, \cdots, \alpha^{2^{m-1}}\}$$

for an element $\alpha \in \mathbb{F}_{2^m}$. One advantage of normal bases is that raising an element to the power of 2 is simply a cyclic shift of the element vector.

2.2 Elliptic Curves

An elliptic curve E over a field $\mathbb{F}$ is defined by a Weierstrass equation:

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

where a_1, a_2, a_3, a_4, $a_6 \in \mathbb{F}$ and the discriminant of E is nonzero. The discriminant of E is defined as follows:

$$\begin{aligned} \Delta &= -d_2^2d_8 - 8d_4^3 - 27d_6^2 + 9d_2d_4d_6 \\ d_2 &= a_1^2 + 4a_2 \\ d_4 &= 2a_4 + a_1a_3 \\ d_6 &= a_3^2 + 4a_6 \\ d_8 &= a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2. \end{aligned} \tag{2.2}$$

For cryptographic applications, elliptic curves are defined over finite fields. In the binary finite field $\mathbb{F}_{2^m}$, i.e., the characteristic is 2, Equation 2.1 can be transformed to

$$E : y^2 + xy = x^3 + ax^2 + b \tag{2.3}$$

where $a, b \in \mathbb{F}$ and $b \neq 0$. The set of points $(x, y) \in \mathbb{F}_{2^m} \times \mathbb{F}_{2^m}$ satisfying Equation 2.3 along with the point at infinity $\mathcal{O}$ forms a group under addition. The point at infinity $\mathcal{O}$ serves as

the identity of the group, i.e., $P + \mathcal{O} = \mathcal{O} + P = P$ for all P on E. For any two points P_1 and P_2 on E, $P_1 \neq \mathcal{O}$, $P_2 \neq \mathcal{O}$, they can be written as $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$.

If $P_2 \neq \pm P_1$, then $P_1 + P_2 = (x_3, y_3)$, where

$$\begin{cases} x_3 = \left(\dfrac{y_1 + y_2}{x_1 + x_2}\right)^2 + \dfrac{y_1 + y_2}{x_1 + x_2} + x_1 + x_2 + a, \\ y_3 = \left(\dfrac{y_1 + y_2}{x_1 + x_2}\right)(x_1 + x_3) + x_3 + y_1 \end{cases} \tag{2.4}$$

The operation is known as point addition.

If $P_2 = -P_1$, then $P_1 + P_2 = \mathcal{O}$.

If $P_2 = P_1$, then $P_1 + P_2 = 2P_1 = (x_3, y_3)$, where

$$\begin{cases} x_3 = {x_1}^2 + \dfrac{b}{{x_1}^2}, \\ y_3 = {x_1}^2 + \left(x_1 + \dfrac{y_1}{x_1}\right) x_3 + x_3 \end{cases} \tag{2.5}$$

The operation is referred to as point doubling.

If the characteristic of the finite field $\mathbb{F}$ is not equal to 2 or 3, the Weierstrass equation can be transformed to

$$E : y^2 = x^3 + ax + b. \tag{2.6}$$

Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$, $P_1 \neq \mathcal{O}$, $P_2 \neq \mathcal{O}$, $P_1 \neq \pm P_2$, point addition $P_1 + P_2 = (x_3, y_3)$ is computed as

$$\begin{cases} x_3 = \lambda^2 - x_1 - x_2, \\ y_3 = \lambda(x_1 - x_3) - y_1 \end{cases} \tag{2.7}$$

where $\lambda = (y_2 - y_1)/(x_2 - x_1)$.

Point doubling $2P_1 = (x_3, y_3)$ is computed as

$$\begin{cases} x_3 = \lambda^2 - 2x_1, \\ \\ y_3 = \lambda(x_1 - x_3) - y_1 \end{cases} \tag{2.8}$$

where $\lambda = (3x_1^2 + a)/(2y_1)$.

Point addition and point doubling consist of arithmetic operations over a finite field such as addition, multiplication, squaring operation and inversion. Inversion over binary finite field or prime field is more expensive than other arithmetic operations. To avoid expensive inversions, affine coordinates can be transformed into projective coordinates. The projective form of the Weierstrass equation of an elliptic curve E can be obtained by replacing x by X/Z^c and y by Y/Z^d where $c, d \in \mathbb{N}$.

In homogeneous projective coordinates, coordinates are transformed by setting $x = X/Z$ and $y = Y/Z$. The equation of elliptic curve over binary finite fields, i.e., Equation 2.3, is transformed to

$$Y^2Z + XYZ = X^3 + aX^2Z + bZ^3. \tag{2.9}$$

An affine point (x, y) is represented by a projective point $(X, Y, Z) = (\theta x, \theta y, \theta)$ for some $\theta \in \mathbb{F}$. The X, Y-coordinate and Z-coordinate can be computed separately for point addition and point doubling.

Affine coordinates can be mapped to projective coordinates in other ways. In the Jacobian projective coordinate, an affine point (x, y) is mapped to a projective point $(X, Y, Z) = (\theta^2 x, \theta^3 y, \theta)$ for some $\theta \in \mathbb{F}$ by setting $x = X/Z^2$ and $y = Y/Z^3$. In the Lopez Dahab projective coordinate (15), an affine point (x, y) is mapped to a projective point $(X, Y, Z) = (\theta x, \theta^2 y, \theta)$ for some $\theta \in \mathbb{F}$ by setting $x = X/Z$ and $y = Y/Z^2$.

2.3 Elliptic Curve Scalar Multiplication

In this section, we introduce the methods to compute ECSM, i.e., kP where k is an integer and P is a point on the elliptic curve E. The ECSM operation dominates the computation in ECC schemes.

2.3.1 Basic Algorithms

The basic algorithms to compute ECSM are binary method and add-and-double-always method. Algorithm 1 shows the left-to-right binary method. The algorithm computes from the second Most Significant Bit (MSB) towards the Least Significant Bit (LSB) of k. It doubles point R for each bit of k and adds P to R only when the bit of k is 1. Therefore, the sequence of operations is dependent on the value of k.

The add-and-double-always method inserts a dummy point addition to make the sequence of operations independent on the value of k, as shown in Algorithm 2.

2.3.2 Montgomery Ladder Algorithm

Montgomery introduced a different method to compute kP (16). He observed that the x-coordinate of the addition of two points can be computed from the x-coordinates of the two points and the x-coordinate of the difference between them. The algorithm computes with two

Algorithm 1 Left-to-right binary method

Input: $P \in E$, $k = (k_{l-1}k_{l-2} \cdots k_0)_2$ where $k_{l-1} = 1$
Output: $Q = kP$

$R \leftarrow P$
for $i = l - 2$ downto 0 **do**
    $R \leftarrow 2R$
    if $k_i = 1$ **then**
        $R \leftarrow R + P$
    end if
end for
Return (R)

Algorithm 2 Add-and-double-always method

Input: $P \in E$, $k = (k_{l-1}k_{l-2} \cdots k_0)_2$ where $k_{l-1} = 1$
Output: $Q = kP$

$R[0] \leftarrow P$
for $i = l - 2$ downto 0 **do**
    $R[0] \leftarrow 2R[0]$
    $R[1] \leftarrow R[0] + P$
    $R[0] \leftarrow R[k_i]$
end for
Return $(R[0])$

points, P_1 and P_2, whose difference is always P after each iteration. In each iteration, one point addition and one point doubling are performed independent on the value of k. Algorithm 3 shows the method.

Algorithm 3 Montgomery ladder algorithm

Input: $P \in E$, $k = (k_{l-1}k_{l-2}\cdots k_0)_2$ where $k_{l-1} = 1$
Output: $Q = kP$
Set $P_1 \leftarrow P$, $P_2 \leftarrow 2P$
for $i = l - 2$ downto 0 **do**
  if $k_i = 1$ **then**
    $P_1 \leftarrow P_1 + P_2$
    $P_2 \leftarrow 2P_2$
  else
    $P_2 \leftarrow P_2 + P_1$
    $P_1 \leftarrow 2P_1$
  end if
end for
Return $(Q = P_1)$

2.3.3 Fast ECSM over $\mathbb{F}_{2^m}$

Lopez and Dahab presented an efficient implementation of the Montgomery ladder algorithm for computing kP on elliptic curves over $\mathbb{F}_{2^m}$ (17). They showed that given $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_2)$ and $P = P_2 - P_1 = (x, y)$, the x-coordinate of $P_1 + P_2$ can be computed as follows.

$$x(P_1 + P_2) = x + (\frac{x_1}{x_1 + x_2})^2 + \frac{x_1}{x_1 + x_2} \tag{2.10}$$

The x-coordinate of the doubling of a point P_1 can also be computed from the x-coordinate of P_1, as shown in Equation 2.5. Therefore, only the x-coordinates of P_1, P_2 and P can be used to perform arithmetic operations needed in the $(l-1)$ iterations in Algorithm 3. At the end of the $(l-1)$th iteration, the x-coordinates of kP and $(k+1)P$ are obtained. The y-coordinate of kP can be recovered as follows.

$$y_1 = (x_1 + x)\{(x_1 + x)(x_2 + x) + x^2 + y\}/x + y \tag{2.11}$$

where x_1, x_2 are the x-coordinates of P_1 and P_2 after the $(l-1)$th iteration.

To avoid expensive inversions over $\mathbb{F}_{2^m}$, the affine coordinates of P_1 and P_2 can be transformed to projective coordinates by setting $x = X/Z$ and $y = Y/Z$. Hence, the x-coordinates of P_1 and P_2, i.e., x_1 and x_2, are represented by X_1/Z_1 and X_2/Z_2.

The x-coordinate of $P_1 + P_2$ can be computed as $\mathrm{X}(P_1 + P_2)/\mathrm{Z}(P_1 + P_2)$ where

$$\begin{cases} Z(P_1 + P_2) = Z_3 = (X_1 \cdot Z_2 + X_2 \cdot Z_1)^2 \\ X(P_1 + P_2) = X_3 = x \cdot Z_3 + (X_1 \cdot Z_2) \cdot (X_2 \cdot Z_1) \end{cases} \tag{2.12}$$

where x is the affine x-coordinate of $P = P_2 - P_1$.

The x-coordinate of $2P_1$ can be computed as $\mathrm{X}(2P_1)/\mathrm{Z}(2P_1)$ where

$$\begin{cases} X(2P_1) = X_4 = X_1^4 + b \cdot Z_1^4 \\ Z(2P_1) = Z_4 = X_1^2 \cdot Z_1^2 \end{cases} \tag{2.13}$$

2.4 Modular Exponentiation

RSA is based on modular exponentiation. Given a message m and an exponent d, modular exponentiation computes $m^d \bmod N$ where N is the modulus.

2.4.1 Algorithms

A basic method for computing modular exponentiation is the binary exponentiation algorithm, also known as the square-and-multiply algorithm. Algorithm 4 shows the left-to-right version which starts at the second MSB of the exponent and works downward. Each bit of exponent is processed with a modular square operation and a conditional modular multiplication. The right-to-left binary exponentiation algorithm is similar to the left-to-right one except that it starts at the LSB and works upward.

Algorithm 4 Binary exponentiation algorithm

Input: m, $d = (d_{l-1} \cdots d_1 d_0)_2$ where $d_{l-1} = 1$, N
Output: $m^d \bmod N$

$R \leftarrow m$
for $i = l - 2$ downto 0 **do**
    $R \leftarrow R^2 \bmod N$
    if $d_i = 1$ **then**
        $R \leftarrow R \cdot m \bmod N$
    end if
end for
Return (R)

In the binary exponentiation algorithm, the sequence of operations is dependent on the value of d since the modular multiplication is performed only when the bit of d is 1. The square-and-multiply-always algorithm inserts a dummy operation to make the sequence of operations independent on d, as shown in Algorithm 5.

Algorithm 5 Square-and-multiply-always algorithm

Input: m, $d = (d_{l-1} \cdots d_1 d_0)_2$ where $d_{l-1} = 1$, N
Output: $m^d \bmod N$

$R[0] \leftarrow m$
for $i = l - 2$ downto 0 **do**
    $R[0] \leftarrow R[0]^2 \bmod N$
    $R[1] \leftarrow R[0] \cdot m$
    $R[0] \leftarrow R[d_i]$
end for
Return $(R[0])$

The Montgomery ladder exponentiation algorithm also ensures the sequence of operations independent on the value of d. As shown in Algorithm 6, it starts at the second MSB of the exponent and processes bit by bit down to the LSB. A modular multiplication and a modular squaring operation are performed in each iteration. $R[1]$ and $R[0]$ satisfy the relation $R[1] = R[0] \cdot m \bmod N$ after each iteration. At the end of the lth iteration, $R[0] = m^d \bmod N$ and $R[1] = m^{d+1} \bmod N$ are obtained. There is no dummy operations in the algorithm.

Algorithm 6 Montgomery ladder exponentiation algorithm

Input: m, $d = (d_{l-1} \cdots d_1 d_0)_2$, N
Output: $m^d \bmod N$

$R[0] \leftarrow 1$, $R[1] \leftarrow m$
for $i = l - 1$ to 0 **do**
    $R[\overline{d_i}] \leftarrow R[\overline{d_i}] \cdot R[d_i] \bmod N$
    $R[d_i] \leftarrow R[d_i] \cdot R[d_i] \bmod N$
end for
Return ($R[0]$)

2.4.2 Computing RSA Fast with Chinese Remainder Theorem

Modular exponentiation in RSA is time-consuming due to the lengths of exponent and modulus. Chinese Remainder Theorem can be used to accelerate computation of the modular exponentiation with the private key in RSA, known as CRT-RSA. Algorithm 7 shows CRT-RSA where $I_q = q^{-1} \bmod p$, $I_p = p^{-1} \bmod q$.

Algorithm 7 CRT-RSA

Input: m, d, p and q where $p \cdot q = N$
Output: $m^d \bmod N$

$d_p \leftarrow d \bmod (p-1)$, $d_q \leftarrow d \bmod (q-1)$
$s_p \leftarrow m_{d_p} \bmod p$
$s_q \leftarrow m_{d_q} \bmod q$
$s \leftarrow \mathrm{CRT}(s_p, s_q) = ((s_p \cdot q \cdot I_q) + (s_q \cdot p \cdot I_p)) \bmod N$
Return (s)

Compared to the straightforward implementation of RSA, CRT-RSA computes two modular exponentiations $m^{d \bmod (p-1)} \bmod p$ and $m^{d \bmod (q-1)} \bmod q$. Since p and q are chosen to have about half the bit length of N, i.e., $|p| \approx |q| \approx |N|/2$, the exponents in the two modular exponentiations are of half bit length of d. Hence, CRT-RSA is about 4 times faster than the straightforward implementation of RSA. Note that CRT-RSA can be used only in digital signature or decryption since it requires to know the secret information p and q.

CHAPTER 3

SIDE CHANNEL ATTACKS AND COUNTERMEASURES FOR PUBLIC KEY CRYPTOSYSTEMS

As introduced in §1.4, side channel attacks exploit the weakness in the implementations of cryptographic algorithms. An attacker can gain information about the internal state of the cryptosystem by monitoring timing, power consumption, electromagnetic emission or erroneous output, and derive the secret key by analyzing the information. Based on the source of leakage, side channel attacks can be divided into timing attacks, power analysis attacks, fault attacks, etc. Fault attacks and power analysis attacks are two of the most powerful ones of all side channel attacks. In this section, we survey fault attacks and power analysis attacks on ECC and RSA and the countermeasures against them.

3.1 Fault Attacks and Countermeasures

Fault attacks are active attacks in which attackers need to disturb the cryptosystem by inducing faults. With erroneous output or the reaction of the cryptosystem under faults, attackers can reveal the secret information. The first fault attack targeted RSA (18) in which a fault was introduced to the computation of one modular exponentiation in CRT-RSA and the secret prime number was derived from the erronous output. Since then, fault attacks have been reported to break other cryptosystems.

3.1.1 Fault Injection and Fault Models

There are many techniques to induce faults. Variations in supply voltage or variations in the external clock may cause data misread, instruction miss or erroneous data. Temperature can lead to misfunctioning of cryptographic device. Circuit manufacturers give the temperature bound for a circuit to function correctly. If the temperature exceeds the bound, operations may not work. And the data stored in memory cells could be modified due to heating. White light can also induce faults due to photoelectric effects. The photons can induce current when a circuit exposed to intense light for a short time period. Laser improves the precision of fault injection. Its directionality allows it to target a small circuit area. X-rays and ion beams can inject faults without depackaging the chip.

The difficulty in inducing a fault depends on the precision in terms of timing and location. Different fault attacks are based on different fault models which require certain control on timing and location. Without any control, a fault is induced randomly. With better control of location, a block of operations can be targeted. Powerful attackers may be able to inject a fault to selected bits with precise control. Attackers may also be able to control the timing of fault injection, however, it is hard to induce a fault at some exact time. Based on the control level of timing, location and the number of bits affected, the most popular fault models used in the literature of fault attacks are single bit fault model, byte fault model, random fault model and arbitrary fault model. The single bit fault model assumes that a fault affects only a single bit of a targeted variable. It is considered unrealistic to target a specific bit of the variable at a specific point of time. Boneh, Demillo and Lipton introduced the single bit fault model

which relaxed the control over location and timing in the first fault attack (18). In the model, a variable used in the cryptographic algorithm rather than a specific bit is targeted, and one bit of the affected variable is flipped. The byte fault model represents the attacks where only a bounded number of bits are affected by an induced fault. When cryptographic devices store and load variables in blocks, the block of bits, typically one byte, can be affected by induced faults. In the byte fault model, a specific variable is targeted, and a fault can affect one byte at an unknown position in the variable. The number of affected bits is usually bounded by a small number, which allows an attacker to try all possible error patterns to identify the correct error pattern for secret information recovery. The random fault model assumes that a variable affected by an induced fault is changed to some random value. It models the case that an attacker lacks the knowledge of how an induced fault affects the value of the variable. The resulting random values are usually assumed to be distributed uniformly to simplify analysis. The weakest fault model is the arbitrary fault model. In the model, an attacker has even less control over location. Only a specific line of the code of cryptographic algorithm can be targeted. And the attacker may not know the effect of the faults, e.g., the distribution of error. An attack based on the arbitrary fault model is more dangerous compared to attacks based on other fault models since it requires less about the capability of an attacker.

3.1.2 Fault Attacks on ECC and RSA

Fault attacks on ECC can be divided into three categories: weak curve based fault attacks, differential fault attacks and safe error attacks.

Weak curve based attacks try to move the computation of ECSM from a strong elliptic curve to a weak elliptic curve. Biehl, Meyer and Muller presented a weak curve based fault attack (19). They observed that parameter a_6 in the Weierstrass equation (Equation 2.1) was not used in computation of ECSM. An attacker can cheat the cryptosystem with a point $P' \in E'$ where E' is a cryptographically weak elliptic curve.

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

The ECSM computed by the cryptographic device for elliptic curve E will generate a point on the elliptic curve E', which differs from E in only a_6. On the weak elliptic curve P' has a smaller order r. Consequently, the attacker can solve ECDLP in a subgroup to retrieve k_r where $k_r = k \bmod r$. By collecting different pairs of k_r and r, k can be derived using Chinese Remainder Theorem. Ciet and Joye then extended the idea and showed that random errors in either coordinates of the base point P, or the elliptic curve parameters or the field representation can help reveal the secret information (20). Fouque et al. presented an attack based on quadratic twist curve (21). They observed that the twist curves of most strong curves are weak. For an elliptic curve E defined over prime field, its twist curve $\tilde{E}$ is defined as

$$\tilde{E} : (\varepsilon)y^2 = x^3 + ax + b \tag{3.2}$$

where ε is a quadratic non-residue in the prime field. According to the curve equation, a random x-coordinate corresponds to a point on either E or $\tilde{E}$. When ECSM is computed with only

x-coordinate, a random fault on the x-coordinate of a point on E has a probability of one half to move to a point on $\tilde{E}$.

Differential fault attacks derive the secret information by analyzing the difference between correct result and erroneous result. Biehl, Meyer and Muller presented differential fault attacks on ECC (19). In the right-to-left binary algorithm for computing ECSM, an attacker can reveal k from the most significant bits to the least significant bits. Every time partial bits of k are targeted. Computing ECSM once, the attacker can obtain the correct result Q. In the second time of computation, the attacker induces a one-bit flip fault on the intermediate result Q'_i after iteration i and obtains the erroneous output Q'. All possible values of targeted bits of k are searched. The correct value is the one which leads to Q_i and Q'_i differing in only one bit. Similarly, the left-to-right binary algorithm (Algorithm 1) can be attacked. Blömer et al. presented a sign change fault attack (22). It targets ECC over prime field. A fault is induced to change a point P to $-P$. This can be achieved by changing the sign of y-coordinate of P. Even though the sign of a point is changed, it is still a valid point on the elliptic curve.

Weak curve based attacks and differential fault attacks need the erroneous result to reveal the secret information. In contrast, safe error attacks do not require to obtain the erroneous results. They simply need the fact that whether the output is affected by induced faults. Yen and Joye presented an attack on ECC based on safe error (23). Faults are induced into the dummy point addition in the add-and-double-always algorithm (Algorithm 2). Whether the dummy point addition will affect the final result depends on the bit value of k, so an attacker can recover k bit by bit by checking the correctness of output.

Boneh, Demillo and Lipton presented the first fault attack on RSA (18). In CRT-RSA, two modular exponentiations are computed with prime p and q, where p and q are the factors of modulus N, i.e., $n = p \cdot q$. An attacker induces a fault during the computation of one modular exponentiation (i.e., s_p or s_q), and an erroneous result s' is produced after CRT combination. With the erroneous signature s' and a correct signature s, the attacker can easily reveal the secret prime number (q or p) by computing Greatest Common Divider of $s' - s$, i.e., $GCD(s' - s)$. The attack assumes that the message padding function is deterministic. Coron extended it to attack RSA when the message is partially unknown due to padding or message encoding (24). The attack finds small roots of linear equations modulo an unknown factor p of n. The straightforward implementation of RSA without CRT is also vulnerable (18). An attacker asks to sign a serial of messages $m_1, m_2, ...m_l$. After collecting erroneous signatures $\hat{s_i}$, the attacker can use $\{m_i, \hat{s_i}\}$ to deduce the secret information. It is assumed that an error occurs at a random iteration during computation of modular exponentiation and flips one bit of the intermediate result. The secret exponent can be recovered in a similar way to differential fault attacks on ECC. A block of bits in secret exponent is recovered at a time. The attacker tries all possible bit vectors until the correct one, which leads to the erroneous output, is found.

Similar to safe error attacks on ECC, safe error attacks on RSA can break the implementation of square-and-multiply-always algorithm (23). In the square-and-multiply-always algorithm (Algorithm 5), modular multiplication is a dummy operation (i.e., the result is discarded) if the value of key bit is 0. If a fault is induced into the dummy modular multiplication, it will

not affect the result of modular exponentiation. Hence, the value of key bit can be revealed by checking whether the result is affected by the induced fault.

3.1.3 Countermeasures against Fault Attacks

In fault attacks, an attacker has to induce faults into the cryptographic device to obtain erroneous result. The erroneous result leaks secret information. Hence, fault attacks can be thwarted by preventing the cryptographic device from outputting erroneous results, which can be achieved by error detection.

Various error detection techniques have been proposed to thwart fault attacks. Elliptic curve integrity check can prevent the fault attacks which move ECSM computation to weak elliptic curves (20) (21). Point verification, which verifies whether a point is on the elliptic curve, is effective against the fault attack presented by Biehl et al. (19). Dominguez-Oviedo et al. showed that point verification misses some errors and they presented error detection methods using input randomization to ensure the correctness of computation of ECSM (25).

Error detection based countermeasures are also developed for RSA. Shamir proposed an approach for CRT-RSA (26). It chooses a random prime number t and computes $s_{pt} = m^{d \bmod (p-1)(t-1)} \bmod p \cdot t$ and $s_{qt} = m^{d \bmod (q-1)(t-1)} \bmod q \cdot t$. And $s_{pt} \equiv s_{qt} \bmod t$ is checked before combining them using CRT. The computation is deemed error free if $s_{pt} \equiv s_{qt} \bmod t$. This method leaves the CRT combination step unprotected. Exploiting this drawback, Aumuller et al. broke the countermeasure proposed by Shamir, and they proposed an improved scheme which also protects the CRT combination step (27). But the random number generation is a problem in the scheme, since generating the random numbers for each signature

operation results in large time overhead. Later, Vigilant presented a scheme to improve it (28). The scheme is still based on modulus extension. It computes $m^d \bmod N$ in Z_{Nr^2} where r is a small random number coprime with N. The message m is transformed to m' which satisfies $m' = m \bmod N$ and $m' = 1 + r \bmod r^2$. Then s and s' are computed as $s = m^d \bmod N$ and $s' = m^d \bmod Nr^2$. Error is detected by verifying the equation $s' \equiv s \bmod N$. The method can be applied to RSA with CRT. However, its performance in terms of time and hardware overhead is not improved much. Giraud proposed a countermeasure for the Montgomery ladder exponentiation algorithm (Algorithm 6) (29). The scheme takes advantage of the relation between the two intermediate results to perform coherence check.

A class of countermeasures based on "fault infective" computation has been proposed (30) (31). The main idea is that if a fault is induced at any step of the algorithm, the final result will be changed in a way that is useless to the attacker.

The safe error attack proposed in (32) does not rely on the analysis of erroneous output. Knowing whether an induced fault affects the output may be enough to reveal the secret information, and such knowledge can be obtained by monitoring output or error signal generated by error detection circuit. This puts in danger all above error detection based schemes that use a check procedure. The "fault infective" based countermeasures may suffer the same problem in the face of the safe error attack since an error occurrence can be verified by simply comparing the output with the correct one. However, a closer investigation shows that the attack is quite implementation-specific and can be thwarted by slightly modifying the implementation.

3.2 Power Analysis Attacks and Countermeasures

Power analysis is a powerful technique to attack a cryptosystem by analyzing the power consumption. The power consumed at a given time during cryptographic computation is related to the data being processed. Kocher et al. described the first power analysis attack on DES (33). The idea then was extended to attack public key cryptosystems ECC and RSA.

Power consumption enables to identify some features in the cryptographic algorithms such as loop, block of operations. If these operations depend on the secret key, the value of the secret key can be recovered by distinguishing the operations. Simple Power Analysis (SPA) is such an attack that distinguishes the key-dependent operations by observing the power consumption of cryptographic device.

Differential Power Analysis (DPA) is more complicated and more powerful than SPA. It exploits the correlation between intermediate cryptographic data and power consumption, and performs statistical analysis on power consumption traces to test hypothetical value of the secret key. DPA usually requires a large number of power consumption traces to recover the secret key. The variants of DPA are presented to attack ECC and RSA by taking advantage of special properties of cryptographic algorithms.

3.2.1 Power Analysis Attacks on ECC and RSA

A basic implementation of ECSM can be vulnerable to SPA. The binary method (e.g., the left-to-right binary method in Algorithm 1) performs operations depending on the value of secret key. Point addition is performed only when the value of key bit is 1. Hence, identifying point addition by power consumption will reveal the key bit.

Coron presented a DPA attack on ECC (34). It monitors the power consumption of smart-card ECC implementation and recovers the secret key stored inside the smart-card. It is assumed that an attacker can ask to compute ECSM with distinct $P_1,\ P_2, \cdots, P_k$ to obtain $Q_1 = dP_1, Q_2 = dP_2, \cdots, Q_k = dP_k$, and monitor the power consumption of every execution. The attacker makes a hypothesis on the value of key bit and derives the intermediate point aP during the computation of ECSM based on the hypothesis. The correlation between the intermediate point aP and power consumption is computed. At time $t = t_1$ when aP is computed, the correlation function will present a "peak" if the hypothesis is correct, otherwise, no "peak" will be observed. Later Goubin presented a Refined Power Analysis (RPA) approach that can successfully attack even if some countermeasures are used (35). The approach makes use of a "special" point P_0, $P_0 \neq \mathcal{O}$, such that one of its coordinates equals zero. The DPA countermeasures based on randomization will not affect the special property of P_0. Hence, correlation analysis for DPA still works. RPA is later extended to Zero Power Analysis (ZPA) by Akishita and Takagi (36). They observed that even if a point had no zero-value coordinate, the registers might take zero-value. ZPA exploits the zero-value registers that can result from certain points to perform statistical analysis. Both RPA and ZPA can be considered as variants of DPA. They assume that the attacker can choose the base point P and ask the cryptographic device to perform ECSM with the fixed secret key, so they are not applicable to ECDSA.

Messerges et al. presented three power analysis scenarios on RSA (37). The three attacks are "Single-Exponent, Multiple-Data" (SEMD) attack, "Multiple-Exponent, Single-Data" (MESD) attack and "Zero-Exponent, Multiple-Data" (ZEMD) attack, with different assumptions of at-

tacker's ability to manipulate the cryptographic device. In the attacks, the power consumption traces are averaged to reduce noise. Then the power consumption of exponentiation with a known exponent is compared to the power consumption of exponentiation with the secret exponent to reveal the secret exponent. Boer et al. proposed a DPA attack on CRT-RSA (38). The attack focuses on the modular reduction performed prior to the modular exponentiations. It makes hypotheses on the remainder after the modular reduction with one of the primes (p or q) and performs correlation on series of power consumption traces of chosen-message RSA to find the prime. Witteman et al. presented a correlation attack by exploiting the relation between consecutive modular squaring operations and modular multiplications (39).

DPA requires a large number of power consumption traces to do statistical analysis, however, Comparative Power Analysis (CPA) can reveal the secret key with only several power consumption traces. CPA assumes that an attacker can input user-defined message to RSA device for computation. The doubling attack, presented by Fouque and Valette, asks for modular exponentiations with input $X \pmod N$ and input $X^2 \pmod N$, respectively (40). Due to the relation between two inputs, a collision will be generated between the power consumption trace of exponentiating X and the power consumption trace of exponentiating X^2 at adjacent operations when the value of key bit is 0. Yen et al. presented an attack using messages $X \pmod N$ and $-X \pmod N$ for modular exponentiations (41). This attack reveals the secret key by taking advantage of the collision which will be observed between the two power traces at modular squaring operations if the value of key bit is 0. Homma et al. generalized the technique (42). They proposed to generate a collision using a message pair (Y, Z) which satisfies

$Y^{\alpha} \equiv Z^{\beta} \pmod{N}$ and detect the collision between two power consumption traces at a certain location determined by α and β. More than one pair of inputs may be needed to reveal the secret key.

3.2.2 Countermeasures against Power Analysis Attacks

SPA can be prevented by making operation sequence independent on the secret information. The add-and-double-always algorithm (Algorithm 2) and the Montgomery ladder algorithm (Algorithm 3) perform one point addition and one point doubling in each iteration independent on the value of k to compute kP. Hence, they are resistant to SPA. For RSA, the square-and-multiply-always algorithm (Algorithm 5) and Montgomery ladder exponentiation algorithm (Algorithm 6) are resistant to SPA. But the add-and-double-always algorithm and the square-and-multiply-always algorithm are vulnerable to the safe error attack due to the dummy operations.

To thwart DPA, techniques of hiding and masking are used. Hiding reduces the correlation between the intermediate data and power consumption by adding noise, reducing signal leakage or using a variable clock frequency. Masking makes the intermediate data unpredictable, thereby breaking the correlation between the intermediate data and power consumption.

Masking can be achieved by randomizing the intermediate data. Countermeasures based on randomization have been proposed to thwart DPA and its variants for ECC. Random scalar splitting computes kP as $kP = rP + (k - r)P$ with a random number r. An alternative is to multiplicatively split k such that $k = \lfloor k/r \rfloor \cdot r + k \bmod r$ in order to reduce the size of r. The scalar k can also be randomized by computing kP as $kP = (k + r\#E)P$, where r is a random

number and $\#E$ is the total number of points on the elliptic curve. The base point P can be blinded by adding a secret random point R to P (i.e., $k(R+P)$) and then subtracting kR from it. Random field isomorphisms and random EC isomorphisms randomize the computation by using an elliptic curve isomorphism and isomorphic representation of the field, respectively. The projective coordinate can be randomized. A point $P = (X, Y, Z)$ is randomized to an equivalent representation $(\lambda X, \lambda Y, \lambda Z)$ where λ is a random number. Mamiya et al. proposed the Binary Expansion with a Random Initial Point algorithm to thwart DPA and RPA/ZPA (43). The algorithm was later improved by Kim et al. (44) and Wang et al. (45).

Messerges et al. suggested to use message masking to prevent MESD and ZESD and use exponent masking to prevent SEMD (37). Prior to modular exponentiation, the message m can be masked with a random value r. The mask is removed after modular exponentiation by multiplying $(r^{-1})^e \bmod N$. Kocher presented an efficient way to compute it (46). To mask the exponent, a random multiple of $\varphi(N)$ can be added to it, i.e., $\hat{e} = e + r\varphi(N)$, where $\varphi(N) = (p-1)(q-1)$. It can be proved that $m^e \equiv m^{\hat{e}} \bmod N$. Messerges et al. described a way to randomize the exponentiation algorithm to protect against power analysis attack (37). The idea is to select a random starting point in the exponent to begin modular exponentiation. This can be achieved by combining the left-to-right binary exponentiation algorithm (Algorithm 4) and the right-to-left binary exponentiation algorithm. The computation of modular exponentiation proceeds from the random starting point towards the MSB using the right-to-left binary exponentiation algorithm, returns to the starting point and then moves towards the LSB using the left-to-right binary exponentiation algorithm. Masking by randomization is also an effective method to

thwart CPA, however, proper random number updating technique should be used, as discussed in (42).

CHAPTER 4

A NEW COUNTERMEASURE AGAINST FAULT ATTACKS FOR ECC

In this chapter, we present LOEDAR, a novel low-cost error detection and recovery scheme for ECC over $\mathbb{F}_{2^m}$. The scheme is based on the Montgomery ladder algorithm (Algorithm 3). We construct an invariant in the Montgomery ladder algorithm. Error is detected by verifying the invariant. The proposed scheme preserves the features in the Montgomery ladder algorithm to retain the fast computation of ECSM discussed in §2.3.3. The scheme also supports fast recovery upon error detecting.

4.1 The Proposed Low-cost Error Detection and Recovery Scheme

The point-updating pyramid of the Montgomery ladder algorithm (Algorithm 3) is shown in Figure 1. It illustrates how points P_1 and P_2 are updated in the first three iterations in the *for* loop of Algorithm 3 that traverses from the MSB to the LSB. At iteration i, the ECSM process will take either the left path (if $k_i = 0$) or the right path (if $k_i = 1$). This is repeated until the LSB k_0 is processed. For example, the path shown in bold arrows is taken when $k = 11 = (k_3k_2k_1k_0)_2 = (1011)_2$.

Figure 1 shows that $P_2 = P_1 + P$ is true at the end of each iteration of the *for* loop. This property of the Montgomery ladder algorithm helps reduce the complexity of computation significantly as discussed in §2.3.2. The computation for ECSM over $\mathbb{F}_{2^m}$ can be further reduced using the method proposed by Lopez and Dahab (17), as discussed in §2.3.3.

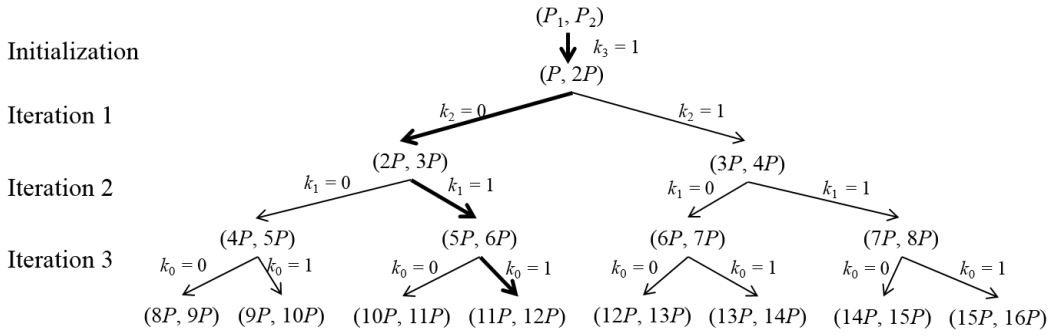

Figure 1. Point updating graph of the Montgomery ladder algorithm

4.1.1 The Idea

In the Montgomery ladder algorithm (Algorithm 3), the difference between P_2 and P_1 is equal to P if no error occurs. Intuitively, the relation $P_2 = P_1 + P$ can be verified at the end of an iteration to detect errors. This idea, however, does not work well due to the following reason. The ECSM is computed fast in projective coordinate using Equation 2.12 and Equation 2.13. So one may want to verify $P_2 = P_1 + P$ by computing $P_1 + P$ in projective coordinate using Equation 2.12 and then comparing the result with P_2. However, adding two points using Equation 2.12 requires the knowledge of x-coordinate of the difference between the two points, i.e., $x(P_1 - P)$ in this case. Since P_1 changes after every iteration during the computation of ECSM, it is impossible to know $x(P_1 - P)$ without adding extra computation to track it. Hence, one cannot verify $P_2 = P_1 + P$ in projective coordinate using Equation 2.12. Verifying $P_2 = P_1 + P$ in affine coordinate using Equation 2.4 is also a choice. In order to add P_1 and P using Equation 2.4, one needs to know both x-coordinate and y-coordinate of the two

points. But ECSM is computed iteratively using Equation 2.12 and Equation 2.13 in which only X-coordinate and Z-coordinate are involved. Coordinate recovery and transformation from projective coordinate (X, Z) to affine coordinate (x, y) are necessary before adding P_1 and P using Equation 2.4. Converting (X, Z) to (x, y) requires the expensive modular inversions, and hence it is quite time-consuming. Moreover, adding two points using Equation 2.4 cost much more than adding two points in projective coordinate using Equation 2.12. Therefore, verifying $P_2 = P_1 + P$ in affine coordinate using Equation 2.4 is not a good choice either.

In order to take the advantage of computing point addition and point doubling using Equation 2.12 and Equation 2.13 in the projective coordinate, we introduce a new point called Verification Point, P_v, such that ECSM as well as computations for error detection can be performed in projective coordinate using Equation 2.12 and Equation 2.13. As shown in Figure 2, P_v is initialized to $\mathcal{O}$ and is incremented at each iteration just like P_1 and P_2. We denote by P_1^i, P_2^i, P_v^i the value of points P_1, P_2, P_v after the ith iteration, respectively. During the ith iteration, P_v is incremented by either P_1 or P_2 depending on the value of k_{l-1-i}, i.e., $P_v^i = P_v^{i-1} + P_1^{i-1}$ if $k_{l-1-i} = 0$, or $P_v^i = P_v^{i-1} + P_2^{i-1}$ if $k_{l-1-i} = 1$, where l is the length of k. For example, the path shown in bold arrows is taken when $k = 11 = (k_3k_2k_1k_0)_2 = (1011)_2, P_v^1 = P_v^0 + P_1^0 = P$ since $k_2 = 0, P_v^2 = P_v^1 + P_2^1 = 4P$ since $k_1 = 0$, and $P_v^3 = P_v^2 + P_2^2 = 10P$ since $k_0 = 1$. Lemma 4.1.1 shows that after the ith iteration $P_v^i + P_2^i = 2P_1^i$ always holds if no error occurs.

Lemma 4.1.1. *If P_v is updated such that $P_v^i = P_v^{i-1} + P_1^{i-1}$ if $k_{l-1-i} = 0$, $P_v^i = P_v^{i-1} + P_2^{i-1}$ if $k_{l-1-i} = 1$, where i indicates the ith iteration and l is the length of k, then $P_v^i + P_2^i = 2P_1^i$ for $1 \leq i \leq l - 1$.*

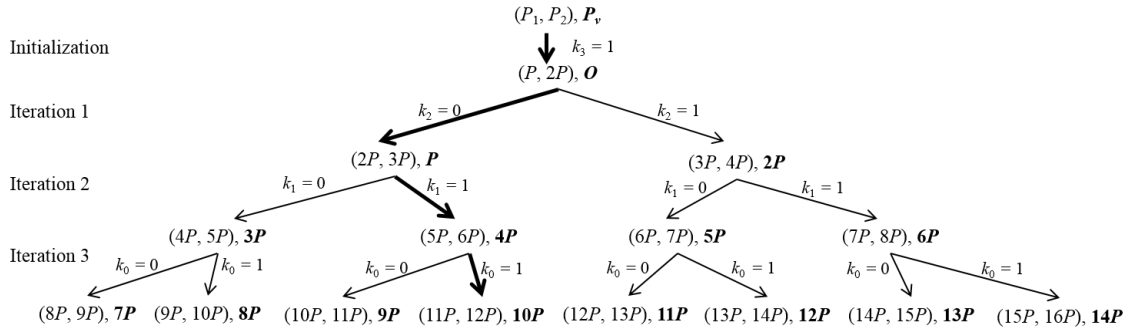

Figure 2. Point updating graph of the LOEDAR scheme

Proof. P_1^0, P_2^0, and P_v^0 are initialized to P, $2P$, and $\mathcal{O}$ respectively where P is the base point on the elliptic curve. After the 1st iteration, P_1, P_2 and P_v are updated as follows.

$$P_1^1 = \begin{cases} 2P & \text{when} \quad k_{l-2} = 0 \\ 3P & \text{when} \quad k_{l-2} = 1 \end{cases} \tag{4.1}$$

$$P_2^1 = \begin{cases} 3P & \text{when} \quad k_{l-2} = 0 \\ 4P & \text{when} \quad k_{l-2} = 1 \end{cases} \tag{4.2}$$

$$P_v^1 = \begin{cases} O + P = P & \text{when} \quad k_{l-2} = 0 \\ O + 2P = 2P & \text{when} \quad k_{l-2} = 1 \end{cases} \tag{4.3}$$

Hence $P_v^i + P_2^i = 2P_1^i$ holds for $i = 1$ no matter $k_{l-2} = 0$ or 1.

Now assume that $P_v^q + P_2^q = 2P_1^q$ holds for $i = q \geq 1$, and we will show that $P_v^{q+1} + P_2^{q+1} = 2P_1^{q+1}$ always holds:

$$P_v^{q+1} = \begin{cases} 2P_1^q & \text{when} \quad k_{l-q-2} = 0 \\ P_1^q + P_2^q & \text{when} \quad k_{l-q-2} = 1 \end{cases} \tag{4.4}$$

$$P_2^{q+1} = \begin{cases} P_1^q + P_2^q & \text{when} \quad k_{l-q-2} = 0 \\ 2P_2^q & \text{when} \quad k_{l-q-2} = 1 \end{cases} \tag{4.5}$$

and the verification point

$$P_v^{q+1} = \begin{cases} P_v^q + P_1^q & \text{when} \quad k_{l-q-2} = 0 \\ P_v^q + P_2^q & \text{when} \quad k_{l-q-2} = 1 \end{cases} \tag{4.6}$$

When $k_{l-q-2} = 0$, we have $P_v^{q+1} + P_2^{q+1} = P_v^q + P_1^q + P_1^q + P_2^q = 4P_1^q = 2P_1^{q+1}$. When $k_{l-q-2} = 1$, we have $P_v^{q+1} + P_2^{q+1} = P_v^q + P_2^q + 2P_2^q = 2(P_1^q + P_2^q) = 2P_1^{q+1}$. Thus $P_v^{q+1} + P_2^{q+1} = 2P_1^{q+1}$. This proves Lemma 4.1.1.

□

4.1.2 The LOEDAR Scheme

The proposed LOEDAR scheme is shown in Algorithm 8. Algorithm 8 has two parts. Part 1 updates P_1 and P_2 just like the Montgomery ladder algorithm (Algorithm 3), which is referred to as the normal ECSM computation. Part 1 also updates P_v concurrently with P_1 and P_2, which is referred to as P_v accumulation. Between two iterations of Part 1, Part 2 verifies the correctness and triggers recovery if errors are detected, and it is referred to as EDR (Error Detection and Recovery). In Part 1, P_1, P_2 and P_v are initialized in the projective coordinate,

and all the operations in Part 1 and Part 2 are performed in the projective coordinate. In EDR errors are detected by verifying if $P_v + P_2 = 2P_1$ is true at the end of an iteration, which involves a point addition that calculates the X and Z coordinates of $P_v + P_2$ (denoted by $X_{P_v+P_2}$ and $Z_{P_v+P_2}$), a point doubling that calculates X and Z coordinates of $2P_1$ (denoted by X_{2P_1} and Z_{2P_1}), and two multiplications $X_{P_v+P_2} \cdot Z_{2P_1}$ and $Z_{P_v+P_2} \cdot X_{2P_1}$ followed by a comparison to verify if $X_{P_v+P_2} \cdot Z_{2P_1} = Z_{P_v+P_2} \cdot X_{2P_1}$. The equation, if holds, implies that $X_{P_v+P_2}/Z_{P_v+P_2} = X_{2P_1}/Z_{2P_1}$ which further implies $P_v + P_2 = 2P_1$. We don't directly compare if $X_{P_v+P_2} = X_{2P_1}$ and $Z_{2P_1} = Z_{P_v+P_2}$ because the transformation of a point from the affine coordinate to the projective coordinate is not unique and different pairs of (X, Z) may map to the same affine x-coordinate. If $X_{P_v+P_2} \cdot Z_{2P_1} = Z_{P_v+P_2} \cdot X_{2P_1}$ is true, no error is detected and P_1, P_2, P_v will be saved as a checked state and then Part 1 is resumed. Otherwise it will rewind to the previous checked state and resume Part 1 from there.

The architecture of the LOEDAR scheme is shown in Figure 3. It is composed of an ECSM/EDR module, an accumulation module, a coordinate transformation module, a point verification module, and a register file. P_v is computed by the accumulation module and updated after each iteration. The ECSM/EDR module updates P_1 and P_2 to compute ECSM and verifies if $P_v + P_2 = 2P_1$ for EDR. At the end of a certain iteration when an EDR process is about to run, the ECSM/EDR module suspends the computation for ECSM and computes for EDR. After finishing all the iterations of ECSM computation, the projective coordinates of P_1 are transformed to affine coordinates by coordinate transformation module, and the result

Algorithm 8 The LOEDAR scheme

Input: An integer $k \geq 0$, and a point $P = (x, y) \in E$
Output: $Q = kP$

Part 1: ECSM and P_v Accumulation
$P_1 \leftarrow P$, $P_2 \leftarrow 2P$, $P_2 \leftarrow \mathcal{O}$
for i from $l - 2$ to 0 **do**
    if $k_i = 1$ **then**
        $P_v \leftarrow P_v + P_2$
        $P_1 \leftarrow P_1 + P_2$
        $P_2 \leftarrow 2P_2$
    else
        $P_v \leftarrow P_v + P_1$
        $P_2 \leftarrow P_2 + P_1$
        $P_1 \leftarrow 2P_1$
    end if
end for
Return $(Q = P_1)$

Part 2: Error Detection and Recovery (EDR)
Pause Part 1 at the end of an iteration
if $P_v + P_2 = 2P_1$ **then**
    save them as a new checked state;
else
    retrieve the previous checked state;
end if
Resume Part 1

will be outputted only after it passes point verification which verifies if a point is on the elliptic curve.

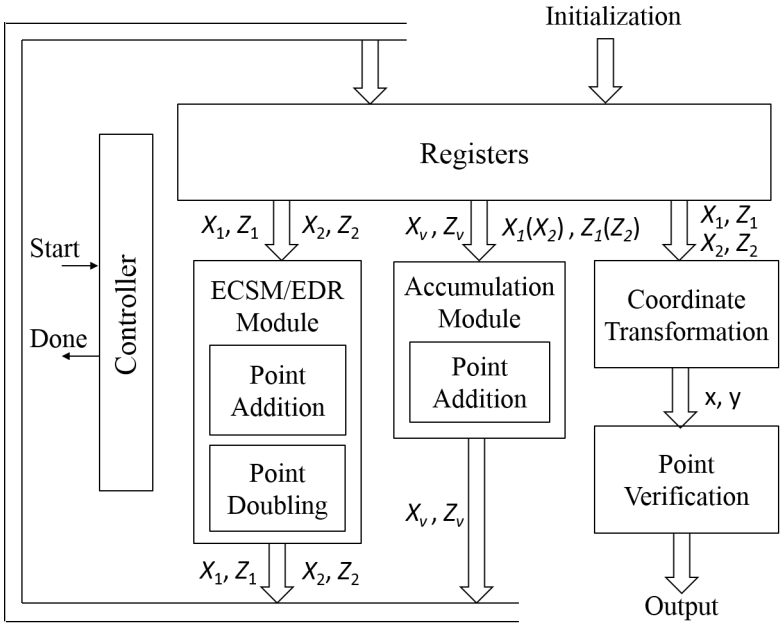

Figure 3. The architecture of the LOEDAR scheme

4.2 Analysis of Error Detection Capability

It is assumed that error occurrences are a random event even though they may be introduced deliberately. This indicates errors may occur anywhere in the computation and any module can be affected by errors with a probability depending on its silicon area. The computation flow is shown in Figure 4. It is assumed that the secret k and the elliplitc curve and field parameters are

loaded, verified, and stored inside the cryptographic device prior to the encryption/decryption or signing/verification process. Besides the secret k, verifying the base point P and other elliptic curve parameters are also necessary since Biehl et al. (19) and Ciet et al. (20) have presented fault attacks by inducing faults to the base point or elliptic curve parameters. The validity of the base point P can be verified by point verification. And the integrity of the elliptic curve parameters can be protected by error detecting code, as suggested in (20).

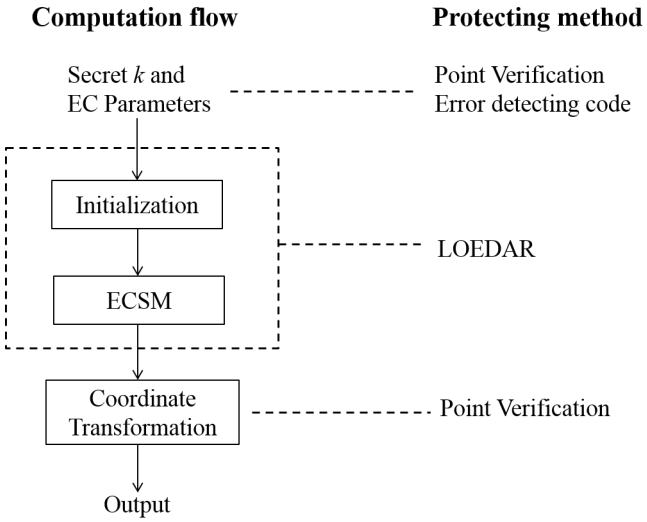

Figure 4. LOEDAR's protection in the computation flow

The initialization step initializes the registers in the datapath using the stored parameters. The ECSM step performs operations in the projective coordinate. The correctness of these two

steps is protected by the proposed LOEDAR scheme. If errors occur during the initialization step such that the computation of ECSM starts with erroneous points P' and $2P'$, errors can be detected by the LOEDAR scheme. The reason is that the point addition shown in Equation 2.12, which uses the x-coordinate of P', is based on the fact that the difference of the two points is P. As a result P' and $2P'$ cannot be added correctly and errors will be detected when $P_v + P_2 = 2P_1$ is verified. If errors occur in the ECSM/EDR process, they will be detected and recovered by the LOEDAR scheme as well with extremely low error missing probability. The detailed analysis is given in this section.

The result of ECSM is then transformed to affine coordinates for output. Errors occurring during coordinate transformation can be detected by point verification and recovered by just transforming the coordinates one more time.

The register file and the FSM controller (not shown in Figure 4) are subject to errors as well. Errors in the registers which are used to store intermediate results for the computation of ECSM and P_v will be detected. Only the ones that store the checked state require special attention. However, they can be protected efficiently using error detecting/correcting codes. The proposed scheme can be implemented with a simple state machine. It can be made fault-tolerant by duplication/triplication with little overhead, or using error detecting/correcting codes with minimum overhead as suggested in (47). Hence the analysis will focus on the errors occurring in the ECSM module and the accumulation module.

The ECSM module and the accumulation module each performs a sequence of operations on $GF(2^m)$ and stores the intermediate data in registers. Errors occurring in the datapath for

any operation will lead to erroneous value to be stored in the corresponding register. Hence, they can be modeled in registers. We assume that an error in the datapath causes an offset to correct value which will be stored in the register. An error occurring in a register storing the intermediate data is also modeled by an offset to the correct value. Since a register can be scheduled to store the outputs of different operations at different clock cycles, an error occurring in a datapath or a register at different time may affect different operations. For simplicity, we ignore register assignments and hardware implementation details and analyze error detection capability based on the Data Flow Graph (DFG). An error occurring in an operation in the DFG can be an error occurring either in the datapath for that operation or in the register storing the output of the operation.

4.2.1 Single Error Occurrence in ECSM/EDR

An error can affect either an operation in the computation for ECSM or an operation in P_v accumulation, or an operation in EDR. While the proposed scheme protects the operations in all the three functions, the capability of detecting errors in the computation for ECSM and P_v accumulation is of the most interest to us since the major goal of the proposed scheme is to deliver correct P_1, P_2 and P_v. The capability of detecting errors in the EDR is of less interest because missing these errors will not affect the correctness of P_1, P_2 and P_v at all. Hence our analysis will focus on the former.

We first consider the case that a single error occurs between two successive EDR processes and the error affects only one operation.

Figure 5 shows the DFG of the point addition and point doubling in ECSM and the point addition in P_v accumulation. The operations in ECSM are numbered from $A1$ to $A7$ for point addition and from $D1$ to $D7$ for point doubling, while the operations of the point addition in the P_v accumulation are numbered from $V1$ to $V7$. Here $\times$, $+$, and S inside circles denote multiplier, adder and squarer, respectively, which perform operations over $\mathbb{F}_{2^m}$. For example, $\times 1$ refers to multiplier 1 and is used four times $(A1, A2, A3, A6)$ for point addition in ECSM. Each of the three functions (i.e., ECSM, P_v accumulation, EDR) is performed on a set of arithmetic units including one multiplier, one adder and one squarer, and they do not share hardware with one another.

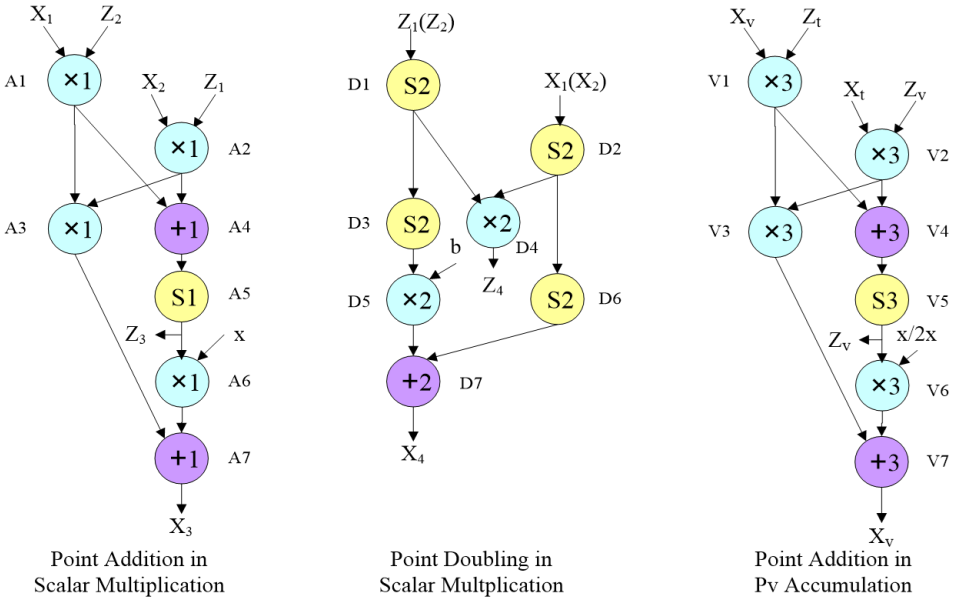

Figure 5. DFG of point addition and point doubling in ECSM and P_v accumulation

An error may occur at any time between two successive EDR processes. A simple scenario is that the error occurs in the iteration right before an EDR process. We will analyze error detection capability in this scenario first, and then show that whenever the error occurs, EDR process can detect it with the same capability. Since the error may occur in any operation, we take $A1$ as an example and assume the error causes an offset O to the output. The offset will be passed through all subsequent operations and will eventually affect the result of point addition. Suppose $P_3 = (X_3, Z_3) = P_1 + P_2$ when no error occurs. The erroneous coordinates are denoted by X_3' and Z_3'.

$$Z_3' = [(X_1Z_2 + O) + X_2Z_1]^2 = Z_3 + O^2 \tag{4.7}$$

$$X_3' = xZ_3 + (X_1Z_2 + O)X_2Z_1 = X_3 + xO^2 + OX_2Z_1 \tag{4.8}$$

If $X_3'/Z_3' = X_3/Z_3$, (X_3', Z_3') is just another mapping of P_3 in the projective coordinate. This is because the mapping of a point from the affine coordinate to the projective coordinate is not unique. The x-coordinate of a point can be mapped to any pair of (X, Z) in the projective coordinate as long as $X/Z = x$. Hence, (X_3', Z_3') can be deemed as a correct result in spite of the error. This type of error has no effect on the correctness of result and can be safely ignored.

If $X_3'/Z_3' \neq X_3/Z_3$, (X_3', Z_3') is not a mapping of the correct P_3. The result is erroneous. Without loss of generality, X_3' and Z_3' can be expressed in Equation 4.9 and Equation 4.10 when an error occurs in any arithmetic unit calculating for the point addition in ECSM.

TABLE III

THE $F_A(O)$ AND $F_B(O)$ OF THE POINT ADDITION OF SCALAR MULTIPLICATION

Erroneous operation	$F_a(O)$	$F_b(O)$
A1	O^2	$xO^2 + OX_2Z_1$
A2	O^2	$xO^2 + OX_1Z_2$
A3	0	O
A4	O^2	xO^2
A5	O	xO
A6	0	O
A7	0	O

$$Z_3' = Z_3 + F_a(O) \tag{4.9}$$

$$X_3' = X_3 + F_b(O) \tag{4.10}$$

where $F_a(O)$ and $F_b(O)$ are functions of offset O and are listed in Table III. Meanwhile the point $P_4 = (X_4, Z_4)$ obtained from the point doubling in ECSM and the point $P_v = (X_v, Z_v)$ from P_v accumulation are not affected by the error.

If the value of the bit of k is 0, the updated P_1 and P_2 are $P_1^U = P_4, P_2^U = P_3$. The EDR process verifies if $P_2^U + P_v = 2P_1^U$, that is $P_3 + P_v = 2P_4$.

$Z'_{P_3+P_v} = Z_{P_3+P_v} + (F_b(O)Z_v + X_vF_a(O)^2$

$X'_{P_3+P_v} = X_{P_3+P_v} + x(F_b(O)Z_v + X_vF_a(O))^2 + X_vZ_v(X_3F_a(O) + F_b(O)Z_3 + F_a(O)F_b(O))$

$Z'_{2P_4} = Z_{2P_4}$

$X'_{2P_4} = X_{2P_4}$

If $Z'_{P_3+P_v}X'_{2P_4} = X'_{P_3+P_v}Z'_{2P_4}$, the error will be missed. This happens only when O satisfies Equation 4.11.

$$\begin{aligned} X_{2P_4}(F_b(O)Z_v + X_vF_a(O))^2 = Z_{2P_4}x(F_b(O)Z_v + X_vF_a(O))^2 \\ +Z_{2P_4}X_vZ_v(X_3F_a(O) + F_b(O)Z_3 + F_a(O)F_b(O)) \end{aligned} \tag{4.11}$$

Consider O as an argument, Equation 4.11 has at most four solutions if the error occurs in operation $A1$, $A2$ or $A4$. Assume O can be any element in $\mathbb{F}_{2^m}$ with equivalent probability. The probability of undetected error is satisfies

$$Pr_{undetected\ error} \leq \frac{4}{2^m - 1} \approx \frac{1}{2^{m-2}} \tag{4.12}$$

If the error occurs in $A3$, $A5$, $A6$ or $A7$, Equation 4.11 has at most 2 solutions. The probability of undetected error satisfies

$$Pr_{undetected\ error} \leq \frac{2}{2^m - 1} \approx \frac{1}{2^{m-1}} \tag{4.13}$$

If the value of the bit of k is 1, the updated P_1 and P_2 are $P_1^U = P_3, P_2^U = P_4$. The EDR process verifies if $P_2^U + P_v = 2P_1^U$, that is $P_4 + P_v = 2P_3$.

$Z'_{P_4+P_v} = Z_{P_4+P_v}$

$X'_{P_4+P_v} = X_{P_4+P_v}$

$Z'_{2P_3} = Z_{2P_3} + Z_3^2(F_b(O))^2 + X_3^2(F_a(O))^2 + (F_a(O)F_b(O))^2$

$X'_{2P_3} = X_{2P_3} + (F_b(O))^4 + b(F_a(O))^4$

If $Z'_{P_4+P_v}X'_{2P_3} = X'_{P_4+P_v}Z'_{2P_3}$, the error will be missed. This happens only when O satisfies Equation 4.14.

$$\begin{gathered} Z_{P_4+P_v}((F_b(O))^4 + b(F_a(O))^4) = \\ X_{P_4+P_v}(Z_3^2(F_b(O))^2 + X_3^2(F_a(O))^2 + (F_a(O)F_b(O))^2) \end{gathered} \tag{4.14}$$

If the error occurs in operation $A1$, $A2$ or $A4$, Equation 4.14 has at most 8 solutions. The probability of undetected error satisfies

$$Pr_{undetected\ error} \leq \frac{8}{2^m - 1} \approx \frac{1}{2^{m-3}} \tag{4.15}$$

If the error occurs in $A3$, $A5$, $A6$ or $A7$, Equation 4.14 has at most 4 solutions. The probability of undetected error satisfies

$$Pr_{undetected\ error} \leq \frac{4}{2^m - 1} \approx \frac{1}{2^{m-2}} \tag{4.16}$$

Similarly, we can compute the probability of undetected error when an error occurs in the point doubling in ECSM or in the point addition in P_v accumulation. The results are summarized in Table IV, where "-" indicates that the probability is independent on the value of the bit of k.

Overall the maximum probability of undetected error is no larger than $\frac{1}{2^{m-3}}$. Since m is usually at least 163 for security reasons, the probability is quite small. As show in Table IV, if

TABLE IV

THE $Pr_{undetected\ error}$ OF THE POINT ADDITION OF SCALAR MULTIPLICATION

Erroneous operation	k_i	$Pr_{undetected\ error}$
A1, A2, A4, D1, D4	0	$Pr_{undetected\ error} \leq \frac{1}{2^{m-2}}$
	1	$Pr_{undetected\ error} \leq \frac{1}{2^{m-3}}$
A3, A5, A6, A7	0	$Pr_{undetected\ error} \leq \frac{1}{2^{m-1}}$
	1	$Pr_{undetected\ error} \leq \frac{1}{2^{m-2}}$
D3, D4, D5, D6, D7	0	$Pr_{undetected\ error} \leq \frac{1}{2^{m-2}}$
	1	$Pr_{undetected\ error} \leq \frac{1}{2^{m-1}}$
V1, V2, V4	-	$Pr_{undetected\ error} \leq \frac{1}{2^{m-2}}$
V3, V5, V6, V7	-	$Pr_{undetected\ error} \leq \frac{1}{2^{m-1}}$

an error occurs in an operation in ECSM, the upper bound of $Pr_{undetected\ error}$ can be different depending on the value of the bit of k, which may cause information leakage. However, it should be quite difficult, if not impossible, to obtain enough samples to exploit the leakage, probabilities as small as $Pr_{undetected\ error}$. Moreover, the upper bound of $Pr_{undetected\ error}$ only gives the worst case of $Pr_{undetected\ error}$. Exploiting its relation to the value of k is more difficult.

Lemma 4.2.1 shows that under transient fault model the incorrect relation among points will be kept over iterations. Hence, the LOEDAR scheme can still detect the error even if the EDR process is not performed right after the iteration where the error occurs.

Lemma 4.2.1. *If in the qth iteration, $P_v^q + P_2^q \neq 2P_1^q$, then in the nth iteration, $q < n < l-1$, inequation $P_v^n + P_2^n \neq 2P_1^n$ is true if no error occurs between the qth iteration and the nth iteration.*

Proof. Given $P_v^q + P_2^q \neq 2P_1^q$ and let $n = q + 1$, we have

$$P_1^{q+1} = \begin{cases} 2P_1^q & \text{when} \quad k_{l-q-2} = 0 \\ P_1^q + P_2^q & \text{when} \quad k_{l-q-2} = 1 \end{cases} \tag{4.17}$$

$$P_2^{q+1} = \begin{cases} P_1^q + P_2^q & \text{when} \quad k_{l-q-2} = 0 \\ 2P_2^q & \text{when} \quad k_{l-q-2} = 1 \end{cases} \tag{4.18}$$

$$P_v^{q+1} = \begin{cases} P_v^q + P_1^q & \text{when} \quad k_{l-q-2} = 0 \\ P_v^q + P_2^q & \text{when} \quad k_{l-q-2} = 1 \end{cases} \tag{4.19}$$

This gives the following:

$$P_v^{q+1} + P_2^{q+1} = \begin{cases} P_v^q + 2P_1^q + P_2^q \neq 4P_1^q & \text{when} \quad k_{l-q-2} = 0 \\ P_v^q + P_2^q + 2P_2^q \neq 2P_1^q + 2P_2^q & \text{when} \quad k_{l-q-2} = 1 \end{cases} \tag{4.20}$$

Hence, after the $(q+1)$th iteration, $P_v^{q+1}+P_2^{q+1} \neq 2P_1^{q+1}$. Now we assume that $P_v^i+P_2^i \neq 2P_1^i$ holds for $n = i$, $q+1 \leq i \leq l-1$, we will show that $P_v^{i+1} + P_2^{i+1} \neq 2P_1^{i+1}$ also holds.

Alfter the $(i+1)$th iteration, we have the following:

$$P_1^{i+1} = \begin{cases} 2P_1^i & \text{when} \quad k_{l-i-2} = 0 \\ P_1^i + P_2^i & \text{when} \quad k_{l-i-2} = 1 \end{cases} \tag{4.21}$$

$$P_2^{i+1} = \begin{cases} P_1^i + P_2^i & \text{when} \quad k_{l-i-2} = 0 \\ 2P_2^i & \text{when} \quad k_{l-i-2} = 1 \end{cases} \tag{4.22}$$

$$P_v^{i+1} = P_v^i + P_t^i \begin{cases} P_v^i + P_1^i & \text{when} \quad k_{l-i-2} = 0 \\ P_v^i + P_2^i & \text{when} \quad k_{l-i-2} = 1 \end{cases} \tag{4.23}$$

This gives the following:

$$P_v^{i+1} + P_2^{i+1} = \begin{cases} P_v^i + P_1^i + +P_1^i + P_2^i \neq 4P_1^i & \text{when} \quad k_{l-i-2} = 0 \\ P_v^i + P_2^i + 2P_2^i \neq 2P_1^i + 2P_2^i & \text{when} \quad k_{l-i-2} = 1 \end{cases} \tag{4.24}$$

This proves Lemma 4.2.1. □

4.2.2 Multiple Error Occurrences in ECSM/EDR

In practice, it is possible that more than one error occurs between two successive EDR processes. But analysis in the case of multiple errors becomes complicated without knowing how many errors occur and which operations they affect. According to Figure 5, errors may affect any operations. Errors may occur in a single iteration or scattered in multiple iterations. When errors affect both ECSM and EDR, the errors in EDR cannot be ignored any longer as they may enable erroneous P_1, P_2 or/and P_v to pass the verification.

Analysis of error detection capability is similar to the single error case. We start with a simple case in which two errors occurs in $A1$ and $A2$, respectively, in the same iteration. Suppose the errors cause offsets $O1$ and $O2$. X_3' and Z_3' are the coordinates of $P_3 = P_1 + P_2$ affected by the errors. According to Figure 5, we have

$$
\begin{aligned}
Z_3' &= (X_1Z_2 + O_1 + X_2Z_1 + O_2)^2 = Z_3 + F_a(O_1, O_2) \\
X_3' &= xZ_3 + (X_1Z_2 + O_1)(X_2Z_1 + O_2) = X_3 + F_b(O_1, O_2)
\end{aligned}
\tag{4.25}
$$

where

$$
\begin{aligned}
F_a(O_1, O_2) &= {O_1}^2 + {O_2}^2 \\
F_b(O_1, O_2) &= x{O_1}^2 + x{O_2}^2 + O_1X_2Z_1 + O_2X_1Z_2 + O_1O_2
\end{aligned}
\tag{4.26}
$$

When the value of the bit of k is 0, the errors will be missed if and only if

$$
\begin{aligned}
&X_{2P_4}(F_b(O_1, O_2)Z_v + X_vF_a(O_1, O_2))^2 = \\
&Z_{2P_4}(x(F_b(O_1, O_2)Z_v + X_vF_a(O_1, O_2))^2 + \\
&X_vZ_v(X_3F_a(O_1, O_2) + F_b(O_1, O_2)Z_3 + F_a(O_1, O_2)F_b(O_1, O_2)))
\end{aligned}
\tag{4.27}
$$

The solutions should be 2-tuples (O_1, O_2). Since O_1 and O_2 are independent on each other and evenly distributed in $\mathbb{F}_{2^m}$, the solution can be obtained by randomly choosing an element in $\mathbb{F}_{2^m}$ for one offset and then making the other offset satisfy Equation 4.27. Hence, the probability of undetected error is

$$
Pr_{undetected\ error} \leq (2^m - 1) \cdot \frac{1}{2^m - 1} \cdot \frac{4}{2^m} \approx \frac{1}{2^{m-2}} \tag{4.28}
$$

Similarly, the probability of undetected error when the value of the bit of k is 1 is

$$
Pr_{undetected\ error} \leq (2^m - 1) \cdot \frac{1}{2^m - 1} \cdot \frac{8}{2^m} \approx \frac{1}{2^{m-3}} \tag{4.29}
$$

The probabilities of undetected error are the same with those in the single error case. In fact, if $n(n \geq 2)$ errors occur and all of them are in the operations of one function, F_a and F_b will become $F_a(O_1, O_2, \cdots, O_n)$ and $F_b(O_1, O_2, \cdots, O_n)$, respectively, where $O_1, O_2, \cdots, O_n$ are n offsets caused by n errors. Since $O_1, O_2, \cdots, O_n$ are independent and evenly distributed, the probabilities of undetected error will be equal to the the worst case of n errors analyzed in the single error scenario.

Errors may occur in the operations of different functions. Without loss of generality, we consider the case that errors occur in every function in an iteration (i.e., point addition in ECSM, point doubling in ECSM and point addition in P_v accumulation) and in the operations of the EDR process as well. Suppose n_1 errors occur in the operations of point addition in ECSM, $n_2 - n_1$ errors in the operations of point doubling in ECSM, and $n_3 - n_2$ errors in the operations of point addition in P_v accumulation. $F_a, F_b, \cdots, F_f$ are the offset functions caused by the errors.

The coordinates of $P_3 = P_1 + P_2$ can be written as

$$\begin{aligned} Z_3' &= Z_3 + F_a(O_1, O_2, \cdots, O_{n_1}) \\ X_3' &= X_3 + F_b(O_1, O_2, \cdots, O_{n_1}) \end{aligned} \tag{4.30}$$

The coordinates of $P_4 = 2P_1$ or $2P_2$ can be written as

$$\begin{aligned} Z_4' &= Z_4 + F_c(O_{n_1+1}, O_{n_1+2}, \cdots, O_{n_2}) \\ X_4' &= X_4 + F_d(O_{n_1+1}, O_{n_1+2}, \cdots, O_{n_2}) \end{aligned} \tag{4.31}$$

The coordinates of P_v can be written as

$$\begin{aligned} Z'_v &= Z_v + F_e(O_{n_2+1}, O_{n_2+2}, \cdots, O_{n_3}) \\ X'_v &= X_v + F_f(O_{n_2+1}, O_{n_2+2}, \cdots, O_{n_3}) \end{aligned} \tag{4.32}$$

If the value of the bit of k is 0, the updated $P_1$and P_2 are $P_1^U = P_4$, $P_2^U = P_3$. The EDR process verifies if $P_2^U + P_v = 2P_1^U$, that is $P_3 + P_v = 2P_4$. The coordinates are

$$\begin{aligned} &Z'_{P_3+P_v} = Z_{P_3+P_v} + G(O_1, O_2, \cdots, O_{n_3}) \\ &X'_{P_3+P_v} = X_{P_3+P_v} + xG(O_1, O_2, \cdots, O_{n_3}) + X_3 Z_v(F_a X_v + F_f Z_3 + F_a F_f) + \\ &\qquad Z_3 X_v(F_b Z_v + F_e X_3 + F_b F_e) + F_h(O_{n_3+1}, O_{n_3+2}, \cdots, O_{n_4}) \\ &Z'_{2P_4} = Z_{2P_4} + (Z_4 F_d + X_4 F_c + F_d F_c)^2 + F_i(O_{n_4+1}, O_{n_4+2}, \cdots, O_{n_5}) \\ &X'_{2P_4} = X_{2P_4} + (F_d)^4 + b(F_c)^4 + F_j(O_{n_4+1}, O_{n_4+2}, \cdots, O_{n_5}) \\ &where \quad G(O_1, O_2, \cdots, O_{n_3}) = (F_a X_v + F_f Z_3 + F_a F_f + F_b Z_v + F_e X_3 + F_b F_e)^2 + \\ &\qquad F_g(O_{n_3+1}, O_{n_3+2}, \cdots, O_{n_4}) \end{aligned} \tag{4.33}$$

F_g, F_h, F_i, F_j are the offset functions when computing $P_3 + P_v$ and $2P_4$ in the EDR process, $n_4 - n_3$ and $n_5 - n_4$ errors occurring, respectively. If $Z'_{P_3+P_v} X'_{2P_4} = X'_{P_3+P_v} Z'_{2P_4}$, the errors will be missed. The solutions should be n_5-tuples $(O_1, O_2, \cdots, O_{n_5})$. Since $O_1, O_2, \cdots, O_{n_5}$ are independent and evenly distributed in $\mathbb{F}_{2^m}$, the solution can be obtained by randomly choosing elements in $\mathbb{F}_{2^m}$ for $n_5 - 1$ offsets and then making the last offset satisfy $Z'_{P_3+P_v} X'_{2P_4} = X'_{P_3+P_v} Z'_{2P_4}$. Therefore, the probability of undetected error will be the worst case of all errors analyzed in the single error scenario. Similarly, we can draw the same conclusion when the

value of bit of k is 1, that is, the upper bound of $Pr_{undetected\,error}$ is $\frac{1}{2^m-3}$ when multiple errors occur in a single iteration.

Now, we show briefly that the upper bound of $Pr_{undetected\,error}$ error is still $\frac{1}{2^m-3}$ when multiple errors are scattered in different iterations in ECSM and/or in the EDR process. The idea is the same as above analysis. The offsets caused by errors will be accumulated over iterations. The solutions of $Z'_{P_3+P_v}X'_{2P_4} = X'_{P_3+P_v}Z'_{2P_4}$ (or $Z'_{P_4+P_v}X'_{2P_3} = X'_{P_4+P_v}Z'_{2P_3}$) can be constructed in the following way. No matter how complicated the functions of offsets caused by the errors are, all offset values except one can be chosen randomly from $\mathbb{F}2^m$. The last offset value should be the one which makes $Z'_{P_3+P_v}X'_{2P_4} = X'_{P_3+P_v}Z'_{2P_4}$ (or $Z'_{P_4+P_v}X'_{2P_3} = X'_{P_4+P_v}Z'_{2P_3}$). As a result, the upper bound of $Pr_{undetected\,error}$ is still around $\frac{1}{2^m-3}$.

4.3 Resistance to Fault Attacks

LOEDAR, as an error detection and recovery scheme, can protect ECC cryptosystem against the fault attacks by detecting and correcting errors. When an error is detected, the erroneous result is prevented from being outputted. Without the erroneous result, an attacker is not able to perform the corresponding analysis to reveal the secret information. The LOEDAR scheme is based on the Montgomery ladder algorithm which does not have dummy operations, hence it can prevent the safe error attacks in which erroneous result is not needed. The protection of the LOEDAR scheme can be further enhanced by integrating curve integrity check into it.

4.4 Extendibility to Thwart Power Analysis Attacks

The LOEDAR scheme is resistant to SPA since it performs the same types of operations, i.e., two point additions and one point doubling, independent on the value of k.

Countermeasures against DPA are essentially based on randomization. Table V summarizes the countermeasures. We can extend the LOEDAR scheme with these countermeasures to thwart power analysis attacks. If the countermeasure can be applied without modifications to the LOEDAR scheme, we say the countermeasure is compatible with the LOEDAR scheme. Table V shows LOEDAR's compatibility with existing countermeasures against DPA and its variants, where "$\surd$" in the column "Compatibility with LOEDAR" indicates that the LOEDAR scheme is compatible with the countermeasure while "$\times$" indicates that the LOEDAR scheme is not compatible with the countermeasure. From Table V, we can see that all the countermeasures listed in the table can help extend LOEDAR to thwart power analysis attacks except the method proposed by Mamiya et al. and its improved versions. The reason is that the compatible countermeasures do not impose any requirement on how to compute ECSM while the method proposed by Mamiya et al. does. Their approach is based on the add-and-double-always algorithm rather than the Montgomery ladder algorithm.

Inspired by the idea of random initial point (43), we present a method for LOEDAR to thwart power analysis attacks by randomizing initial point, as shown in Algorithm 9. In Algorithm 9, the initial points P_1, P_2 and P_v are blinded by a random point R. And a new point P_3, which is doubled in every iteration, can remove the increment caused by R after all iterations. R is a point on the elliptic curve E, i.e., $R = (x_R, y_R) \in E$. The randomness of R is defined by randomness of its coordinates. A simply way to generate R is to choose one coordinate (x_R or y_R) randomly on $\mathbb{F}_{2^m}$ and then compute the other one (y_R or x_R) such that $R = (x_R, y_R) \in E$. If there is no solution, repeat the procedure until a valid point is obtained. A more efficient

TABLE V

LOEDAR'S COMPATIBILITY WITH COUNTERMEASURES AGAINST POWER ANALYSIS

Countermeasures against Power Analysis	Resistance against		Compatibility with LOEDAR
	DPA	RPA/ZPA	
Random scalar splitting (48)	√	√	√
Scalar randomization (34)	√	√	√
Base point blinding (34)	√	√	√
Random field isomorphisms (35)	√	×	√
Random EC isomorphisms (35)	√	×	√
Random projective coordinates (34)	√	×	√
Binary expansion with a random initial point and its improved version (43) (44) (45)	√	√	×
Alogirthm 9	√	√	√

method is discussed in (43). The extended LOEDAR scheme can thwart DPA as well as RPA/ ZPA, since special points or zero-value registers used by RPA and ZPA will be randomized by R.

Besides the proposed extended scheme in Algorithm 9, the countermeasures listed in Table V can be used to extend the LOEDAR scheme. The costs of extending the LOEDAR scheme with the countermeasures in Table V are compared and the result is given in Table VI where "A" indicates point addition, "D" indicates point doubling, and l is the length of k. From Table VI, we can see that extending the LOEDAR scheme with the scalar randomization technique does not require additional point operations. But the random number r should be chosen large

Algorithm 9 The extended LOEDAR scheme

Input: An integer $k \geq 0$, and a point $P = (x, y) \in E$
Output: $Q = kP$

Part 1: ECSM and P_v Accumulation
Choose a random point R
Set $P_1 \leftarrow P + R$, $P_2 \leftarrow 2P + R$, $P_3 \leftarrow -R$, $P_v \leftarrow R$
for i from $l - 2$ to 0 **do**
    $P_3 \leftarrow 2P_3$
    if $k_i = 1$ **then**
        $P_v \leftarrow P_v + P_2$
        $P_1 \leftarrow P_1 + P_2$
        $P_2 \leftarrow 2P_2$
    else
        $P_v \leftarrow P_v + P_1$
        $P_2 \leftarrow P_2 + P_1$
        $P_1 \leftarrow 2P_1$
    end if
end for
$P_1 \leftarrow P_1 + P_3$
Return $(Q = P_1)$

Part 2: Error Detection and Recovery (EDR)
Pauses Part 1 at the end of an iteration
if $P_v + P_2 = 2P_1$ **then**
    save them as a new checked state;
else
    retrieve the previous checked state;
end if
Resume Part 1

enough for security purpose (40). And the randomized scalar should be computed carefully since it may leak useful information to an attacker to mount the carry-based attack (49).

TABLE VI

COMPARISON OF COUNTERMEASURES WHEN APPLIED TO LOEDAR

Countermeasures against Power Analysis	Amount of Operations	Overhead
LOEDAR without countermeasure against power analysis	$2(l-1)A + (l-1)D$	–
Random scalar splitting (48)	$4(l-1)A + 2(l-1)D$	100%
Scalar randomization (34)	$2(l-1)A + (l-1)D$	0
Base point blinding (34)	$4(l-1)A + 2(l-1)D$	100%
Alogirthm 9	$2(l-1)A + 2(l-1)D$	$< 33\%$

4.5 The Experiments

We model the implementation of ECC based on the Montgomery ladder algorithm and the proposed LOEDAR scheme using VHDL and synthesize it into netlist using the Synopsys Design Vision and the TSMC 65nm library. The ECSM/EDR module consists of two multipliers, two squarers and two adders, and the accumulation module consists of one multiplier, one squarer and one adder. All operations are over $\mathbb{F}_{2^{163}}$. We use the MSD(Most Significant Digit)-first digit serial multipliers over $\mathbb{F}_{2^{163}}$ as in (50) with digit size $d = 55$. We implement squarers instead of using multipliers for squaring operations to improve the performance. The squarers are similar to the ones in (51). The divider used in the coordinate transformation module is similar to the one given in (52). The maximum system frequency is above 200MHz.

4.5.1 Overhead

The area and delay of the arithmetic units are reported in Table VII. The hardware overhead of the LOEDAR scheme compared to the regular ECSM based on the Montgomery ladder

algorithm is shown in Table VIII. The LOEDAR scheme costs around 37% hardware overhead. The overhead is introduced by the accumulation module and the registers for intermediate storage in the EDR process.

TABLE VII

THE AREA AND DELAY OF THE ARITHMETIC UNITS

Arithmetic Unit	NAND2-equivalent Area	Time (clock cycles)
Multiplier	27824	4
Squarer	559	1
Adder	408	1
Divider	8914	Depending on inputs

TABLE VIII

HARDWARE OVERHEAD

Schemes	NAND2-equivalent Area	Overhead
Regular ECSM	153127	0
LOEDAR	210696	37.6%

Table IX illustrates the time overhead and the switching activity overhead of the LOEDAR scheme. The time overhead depends on how many times the EDR process is performed. Every time the EDR process is performed, 0.68% time overhead will be introduced. Hence the total time overhead can be expressed by $n \times 0.68\%$ where n is the total number of times the EDR process is performed during the computation of ECSM. The power overhead is reported in the form of average switching activities. We use VSS to simulate the gate-level netlist and capture the average switching activities over dozens of different inputs. The switching activities produced by the LOEDAR scheme when EDR process is not performed are 69% more than the regular ECSM. The overhead is caused by the accumulation module. Every time the EDR process is performed, it results in 0.6% more switching activities.

TABLE IX

TIME OVERHEAD AND SWITCHING ACTIVITY OVERHEAD

Schemes		Time (clock cycles) /Overhead	Average Switching Activities ($\times 10^3$) /Overhead
Regular ECSM		3084/-	166408/-
LOEDAR	LOEDAR w/o EDR	3084 / 0%	281127 / 69%
	Each EDR process	21 / 0.68%	1020 / 0.6%
	LOEDAR w/ EDR	$3084 + 21n$ $/n \times 0.68\%$	$281127 + 1020n$ $/ 69\% + n \times 0.6\%$

Note that we by no means claim the optimality or efficiency of our implementation of the basic ECC architecture, since our major contribution is on developing a novel countermeasure with low cost. As can be seen from the previous sections, the proposed scheme has the potential to work with any architecture based on the Montgomery ladder algorithm, e.g., the architecture in (53).

4.5.2 Comparison to Existing Schemes

In order to compare the LOEDAR scheme with the existing schemes in (25) that support both error detection and recovery, we have ported our implementations to the same FPGA as the one used in (25). In (25), three schemes have been proposed. The TMR(Triple Modular Redundancy)-based ECSM scheme utilizes three ECSM modules to perform the same operation but with inputs encoded by the randomization technique and produces a correct output by a majority voter. In the DMR_PV(Double Modular Redundancy and Point Verification)-based ECSM scheme, two ECSM modules and two point verification modules work in parallel with randomized inputs and their outputs are verified by the point verification modules. The PRC(Parallel and Recomputation)-based ECSM scheme also uses two ECSM modules to compute in parallel with randomized inputs but it improves error detection probability.

The areas of the arithmetic units are compared in terms of slices, and the result is shown in Table X. Our multiplier uses the same architecture as (25).

Table XI compares the areas of the complete implementations. Our scheme uses less area than all the schemes in (25).

TABLE X

COMPARISON OF THE AREAS OF THE ARITHMETIC UNITS

Arithmetic Unit	Area (slices)	
	LOEDAR	Schemes in (25)
Multiplier	2273	2364
Squarer	84	165
Adder	94	94

TABLE XI

COMPARISON OF THE AREAS OF THE COMPLETE SCHEMES

Schemes	Area (slices)
LOEDAR	10636
TMR ECSM (25)	17194
DMR_PV ECSM (25)	12916
PRC ECSM (25)	13136

The comparison of the performance is shown in Table XII. The clock frequency is 52MHz in our implementation and 66MHz in (25). The time required by each scheme can be computed based on the clock frequency and the number of clock cycles needed to finish one ECSM operation. Since every EDR process in the LOEDAR scheme consumes additional 0.4 us, the total time can be expressed by $59.3 + 0.4n$ us where n is the number of times the EDR process is performed during the computation of ECSM.

We can see that the proposed scheme is much faster than the schemes in (25) even if the EDR process is performed after every iteration, i.e., $n = 162$.

TABLE XII

COMPARISON OF PERFORMANCE

Schemes	Freq. (MHz)	Time (us)
LOEDAR w/ EDR	52	$59.3 + 0.4n$
TMR ECSM (25)	66	319.5
DMR_PV ECSM (25)	66	294.7
PRC ECSM (25)	66	290.2

4.5.3 Error Detection and Recovery Test with Random Fault Injection

An attacker may have partial control over the location and timing of fault injection, however, precise control is extremely difficult, if not impossible. In order to pass the verification, an attacker has to solve the equation like Equation 4.11 or Equation 4.27 to obtain the value of the offset and inject a fault to produce the offset. This may require the attacker to precisely control the fault. To the best of our knowledge, existing fault injection techniques cannot achieve it. Since we cannot predict the timing and location of the induced fault or the number of bits affected by the induced fault, we simulate random faults to test the error detection capability. We inject faults to the implementation of the LOEDAR scheme. Due to the scale of hardware and running time we use RTL-level fault injection technique and the bit-flip fault

model. Random faults are injected into the VHDL model in a similar way to (54). The idea is to XOR the content of a target register with a fault control signal of the same length such that randomly selected bits are flipped and the erroneous data will be inputted to the corresponding arithmetic unit for subsequent computations.

We simulate the following two cases. In the first case, we randomly set one bit of fault control signal to 1 while keeping others to be 0. In the second case, we generate a random number as fault control signal using the method described in (55). An iteration is randomly chosen to induce the fault. The result is compared with the one in the fault-free case to test the effectiveness of the LOEDAR scheme. We run the simulation on a computer with 3.4 GHz CPU and 1.8 GB memory for several days. The results show that the LOEDAR scheme has detected errors and recovered successfully in all test cases. The results are expected according to the extremely small probability of undetected error given in §4.2.

CHAPTER 5

A NEW COUNTERMEASURE AGAINST FAULT ATTACKS FOR RSA

In this chapter, we present a novel Concurrent Error Detection (CED) scheme for RSA to thwart fault attacks. An invariant based on the multiplicative homomorphic property is exploited to detect errors. Specifically, the proposed CED scheme verifies if $\Pi_{i=1}^{k} E(m_i) \equiv E(\Pi_{i=1}^{k} m_i \bmod N) \bmod N$ to detect errors, where E can be RSA encryption/decryption or signing/verification process. The scheme has good error detection capability. The probability of missing an error is extremely low. And it does not impose any requirement on how to implement RSA, i.e., the modular exponentiation. It can work with any RSA implementation, e.g., an implementation resistant to power analysis attacks. Hence, it is easy to make the proposed CED scheme thwart other side channel attacks.

5.1 Overview

RSA can be used for encryption/decryption or signing/verification. For simplicity, we will describe the proposed scheme assuming the application of encryption/decryption. More specifically, we explain the scheme in the encryption scenario. In practice, however, the decryption process may be the major concern since the private key is used in the process. The decryption process is the same as the encryption process from the perspective of computation. Both of them compute modular exponentiation in spite of different inputs (base and exponent). Hence, the proposed scheme can protect the decryption process against fault attacks. RSA in

signing/verification is the same as RSA in encryption/decryption except the roles of keys are reversed. The scheme can also be applied in signing/verification.

The proposed CED scheme exploits the multiplicative homomorphic property of RSA to detect errors. As a countermeasure against fault attacks, it has the following advantages over the state-of-art countermeasures:

- It is simple and can work with any RSA implementation, which can be a high-performance implementation that employs Residue Number System, the Montgomery ladder exponentiation algorithm, pipelined datapath, or a fine-tuned one resistant to power analysis attacks, timing attacks, etc. The proposed scheme enables an easy divide-and-concur method to thwart other types of side channel attacks: any fine-tuned RSA implementation, e.g. an implementation resistant to power analysis attacks, can be made resistant to fault attacks by applying the proposed CED scheme. The original properties, e.g. resistance to power analysis attacks, will not be affected. Most of the existing countermeasures against fault attacks, on the other hand, are implementation-specific. Extending them to gain resistance to other types of side channel attacks may result in new leakage.
- Because it works with any RSA implementation, it can be used to protect both the encryption process and the decryption process. The encryption process is of less interest to an attacker since the public key is used in the encryption process. However, the complexity of RSA datapath and the vulnerability of VLSI process technology raise the concern of the reliability against accidental faults due to environment radiations (56). The proposed CED scheme can be used to detect errors in the encryption process when

accidental faults occur. The existing countermeasures designed for CRT-RSA require to know the secret primes p and q. Some of them even require the knowledge of both keys e and d. Such knowledge, however, is often impractical to the encryption process, and hence prevents the existing countermeasures being deployed to detect errors caused by accidental faults.

- It has good error detection capability. With the single bit fault model, breaking the proposed CED scheme is as hard as factoring the modulus N.

In order to achieve high performance, the CED scheme requires to process successive messages with the same key. It may not work efficiently when the RSA cryptographic device constantly switches among sessions that use different keys. The memory required is not negligible, especially in the resource-constrained devices. But it is small compared to the large datapath of RSA and can fit in most of the modern low-end FPGAs.

5.2 The Proposed Concurrent Error Detection Scheme

RSA is based on modular exponentiation which makes it multiplicative homomorphic. An encryption is homomorphic if given $E(m_1)$ and $E(m_2)$ one can obtain $E(m_1 \Theta\ m_2)$ without decrypting m_1 and m_2 for some operation Θ. For example, an encryption is additive homomorphic if $E(m_1) + E(m_2) = E(m_1 + m_2)$, e.g., Paillier encryption (57), and an encryption is multiplicative homomorphic if $E(m_1) \cdot E(m_2) = E(m_1 \cdot m_2)$.

RSA encryption is multiplicative homomorphic since $E(m_1 \cdot m_2) = (m_1 \cdot m_2)^e \bmod N = (m_1^e \bmod N) \cdot (m_2^e \bmod N) = E(m_1) \cdot E(m_2) \bmod N$. This property can be exploited to detect errors.

5.2.1 The Basic CED Scheme

Algorithm 10 shows a basic CED scheme based on the homomorphic property.

Algorithm 10 The basic CED scheme

Input: m_1, m_2, e, N
Output: $c_1 = (m_1)^e \bmod N$, $c_2 = (m_2)^e \bmod N$
$c_1 \leftarrow E(m_1)$
$c_2 \leftarrow E(m_2)$
$c_3 \leftarrow E(m_1 \cdot m_2 \bmod N)$
if $c_3 \equiv (c_1 \cdot c_2) \bmod N$ **then**
    Output(c_1, c_2)
else
    Signal error, suppress (c_1, c_2)
end if

The basic CED scheme encrypts two messages and buffers the results in c_1 and c_2, and then it encrypts the product of the two messages and buffers the result in c_3. According to the multiplicative homomorphic property, $E(m_1 \cdot m_2) = E(m_1) \cdot E(m_2)$, that is, $c_3 \equiv c_1 \cdot c_2 \bmod N$. The relation is verified before outputting c_1 and c_2. A mismatch indicates an error.

The scheme performs one verification for every two encryptions. The time overhead is mainly caused by computing $m_1 \cdot m_2 \bmod N$, $E(m_1 \cdot m_2 \bmod N)$ and $c_1 \cdot c_2 \bmod N$. The modular multiplication can be computed fast and the computing time is negligible compared to the modular exponentiation, i.e., $E(m_1 \cdot m_2 \bmod N)$. Hence, the time overhead can be

estimated by only considering the modular exponentiations. It is approximately 50% for the basic CED scheme.

Due to the verification process, an error may not be detected immediately after it occurs. We define error detection latency as the time delay between the time of error occurrence and the time when it is detected. The error detection latency of the proposed CED scheme, in the worst case, is approximately the time of three encryptions when an error occurs near the beginning of the encryption of m_1. The output latency for either message is increased too, because the results c_1 and c_2 can be outputted only when the verification is passed.

The hardware overhead can be small since encrypting the product of the two messages can be computed in the same datapath as encrypting a single message. Even sharing the datapath, the CED scheme can detect errors in the case of permanent faults, as explained in §5.3. For security purpose, c_1, c_2 and c_3 should be kept unaccessible to an attacker before the verification is passed. Buffering c_1, c_2 and c_3 for verification incurs memory overhead.

When a comparator is used to compare c_3 to the result of $c_1 \cdot c_2 \bmod N$, it is important to protect the comparator. A self-checking comparator (58) (59) or duplicate comparators can be used to detect errors occurring inside the comparator. The duplicate comparators operate in parallel. They are programmed to err on the side of caution and signal an error whenever either of them detects a mismatch.

We illustrate the basic CED scheme using a simple example in which the inputs are quite small, but the proposed CED scheme can be used for RSA with inputs of length of thousands

of bits. Let two primes $p = 61$ and $q = 53$, and $N = p \cdot q = 3233$. Assume message $m_1 = 65$ and message $m_2 = 504$ are encrypted with the key $(N = 3233, e = 17)$. The ciphertexts are

$$c_1 = (m_1)^e \bmod N = 65^{17} \bmod 3233 = 2790$$

$$c_2 = (m_2)^e \bmod N = 504^{17} \bmod 3233 = 866$$

The encryption of the product of m_1 and m_2 is computed as follows.

$$c_3 = (m_1 \cdot m_2 \bmod N)^e \bmod N = (65 \times 504 \bmod 3233)^{17} \bmod 3233 = 1089$$

Since $c_1 \cdot c_2 \bmod N = 2790 \times 866 \bmod 3233 = 1089 = c_3$, the equation $c_3 \equiv c_1 \cdot c_2 \bmod N$ holds.

5.2.2 The Modified CED Scheme

The basic CED scheme shown in Algorithm 10 requires to process two messages at a time. In this section, we present the methods to modify the basic CED scheme for the encryption of a single message.

We consider the case that a single message m_1 is encrypted. To take advantage of the homomorphic property, we generate a second message and compute in the same way as the basic CED scheme. The second message can be generated by a random number generator for every execution or set constant. The corresponding CED schemes are given in Algorithm 11 and Algorithm 12, respectively.

The scheme in Algorithm 11 needs a random number generator, and the time overhead increases to 200%. Using a constant message (Algorithm 12) has several advantages. It saves the hardware of a random number generator and requires only a buffer to hold the constant

Algorithm 11 The modified CED scheme using a random message

Input: m_1, e, N
Output: $c_1 = (m_1)^e \bmod N$

Choose a random number r, $0 < r < N$
$c_1 \leftarrow E(m_1)$
$c_r \leftarrow E(r)$
$c_3 \leftarrow E(m_1 \cdot r \bmod N)$
if $c_3 \equiv (c_1 \cdot c_r) \bmod N$ **then**
    Output(c_1)
else
    Signal error, suppress (c_1)
end if

Algorithm 12 The modified CED scheme using a constant message

Input: m_1, m_{const}, c_{const}, e, N
Output: $c_1 = (m_1)^e \bmod N$

$c_1 \leftarrow E(m_1)$
$c_3 \leftarrow E(m_1 \cdot m_{const} \bmod N)$
if $c_3 \equiv (c_1 \cdot c_{const}) \bmod N$ **then**
    Output(c_1)
else
    Signal error, suppress (c_1)
end if

message m_{const}. The encryption of the constant message, i.e., c_{const}, can be pre-computed to save the time of computing $c_r = E(r)$ in Algorithm 11, and thereby to reduce the time overhead to 100%. But using a constant message may leak information. Existing attacks do not make use of it to the best of our knowledge, though.

5.2.3 The Enhanced CED Scheme

The basic CED scheme incurs 50% time overhead and the modified CED schemes incurs 100% (using a constant message) or 200% (using a random message) time overhead. To reduce the time overhead, we present an enhanced scheme in this section.

Since the multiplicative homomorphic property holds for more than two messages, the verification can be performed for every k messages, $k > 2$. The enhanced CED scheme is given in Algorithm 13.

Algorithm 13 The enhanced CED scheme

Input: $m_1, m_2, \cdots, m_k$, e, N
Output: $c_1 = (m_1)^e \bmod N$, $c_2 = (m_2)^e \bmod N$, $\cdots$, $c_k = (m_k)^e \bmod N$
$c_1 \leftarrow E(m_1)$
$c_2 \leftarrow E(m_2)$
$\vdots$
$c_k \leftarrow E(m_k)$
$c_{k+1} \leftarrow E(\Pi_{i=1}^{k} m_i \bmod N)$
if $c_{k+1} \equiv (\Pi_{i=1}^{k} c_i) \bmod N$ **then**
    Output$(c_1, c_2, \cdots, c_k)$
else
    Signal error, suppress $(c_1, c_2, \cdots, c_k)$
end if

In Algorithm 13, k consecutive messages $m_1, m_2, \cdots, m_k$ are encrypted followed by the encryption of the product of them, i.e., $E(\Pi_{i=1}^{k} m_i \bmod N)$. According to the homomorphic

property, $c_{k+1} \equiv (\Pi_{i=1}^{k} c_i) \bmod N$ holds. If it is not true, an error is detected and the results $c_1, c_2, \cdots, c_k$ will not be outputted.

The scheme reduces the time overhead to $\frac{1}{k}$. The fault detection latency, in the worst case, is approximately the time of $k+1$ encryptions when an error occurs near the beginning of the encryption of the first message. The output latency of each message depends upon its time of arrival. For an application, the acceptable maximum output latency limits the value of k. We call k the depth of the CED scheme. For example, a server responsible for distributing private keys for bulk data encryption under severe time constraints should use a smaller k to meet the individual deadlines, while a server responsible for supplying large size public certificates may opt for a larger k in order to reduce the time overhead. When a message arrives, it can be multiplied into the product of the previous messages. The result is buffered to multiply next message until k messages have been processed. Then the encryption of the product is computed and stored to the buffer. In addition, the ciphertexts of k messages need to be buffered before verification. Hence, the memory overhead for $k > 2$ is increased. It is also a factor limiting the depth of the CED scheme.

5.2.4 One for All: Unifying the CED Schemes

In fact, the basic scheme, the modified scheme and the enhanced scheme can be unified as one, as shown in Figure 6. The depth, i.e., k, can be chosen based on the requirement of the application.

The proposed CED scheme can be applied to the *secure* implementations of RSA which convert a message m to $f(m)$ before the encryption, where f is a nonlinear transformation such

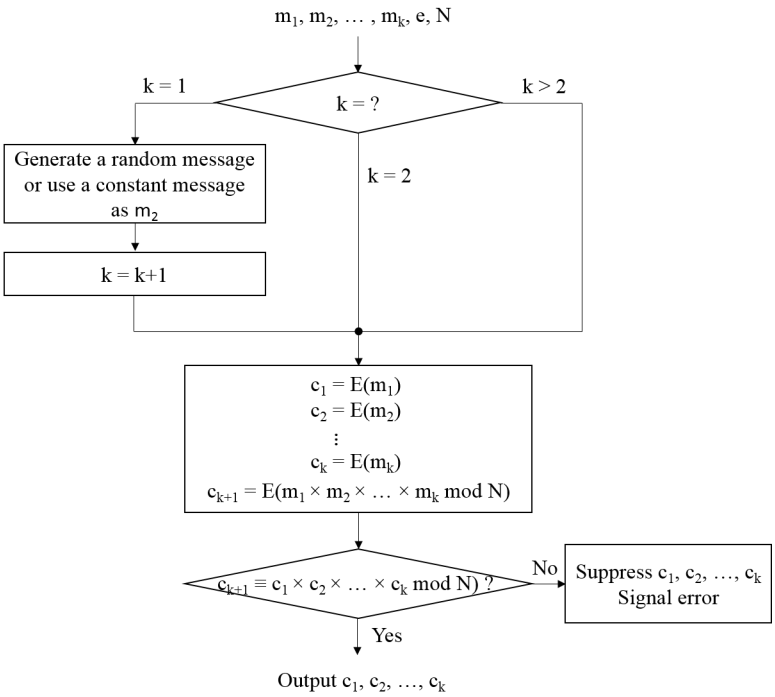

Figure 6. The unified CED scheme

as hash functions based on either PKCS #1 or OAEP (7). For the *secure* implementations of RSA, the proposed CED scheme will check if $\prod_{i=1}^{k} E(f(m_i)) \equiv E(\prod_{i=1}^{k}(f(m_i) \bmod N) \pmod N$ holds to detect errors.

5.3 Error Detection Capability and Resistance to Fault Attacks

There are two sources of faults in a circuit: environment-induced accidental faults and attacker-induced intentional faults. The occurrence and location of accidental faults (e.g., environment radiation induced bit flips) depends on the sensitivity of the circuit nodes to radiation. Without knowing the implementation details of the crpytographc device, accidental

faults are treated as random faults. A fault can occur at any location with the same probability. On the other hand, intentional faults are deliberately induced by attackers to some specific location at some point of time. Hence, we analyze the error detection capability of the proposed CED scheme in the two scenarios.

5.3.1 Intentional Faults

We start the analysis by considering the basic CED scheme ($k = 2$ in the unified CED scheme) and show that the conclusion is also true for the modified CED scheme ($k = 1$ in the unified CED scheme) and the enhanced CED scheme ($k > 2$ in the unified CED scheme).

Lemma 5.3.1. *The basic CED scheme detects errors by verifying if* $c_1 \cdot c_2 \equiv c_3 \bmod N$ *holds, where* $c_1 = (m_1)^e \bmod N$, $c_2 = (m_2)^e \bmod N$, *and* $c_3 = (m_1 \cdot m_2 \bmod N)^e \bmod N$. *Inducing a fault such that the error will not be detected is as hard as factoring the modulus* N.

In order to prove it, we assume that the the error can occur in any one of the four computations. There are two cases:

Case 1: the error occurring during the computation of $c_1 = (m_1)^e \bmod N$ or $c_2 = (m_2)^e \bmod N$.

Case 2: the error occurring during the computation of $c_3 = (m_1 \cdot m_2 \bmod N)^e \bmod N$ or $(c_1 \cdot c_2) \bmod N$.

At the register transfer level, an error occurring in an operator can be modeled as an offset from the correct output value, denoted by O. Assume that an erroneous RSA encryption with message m, encryption key e, modulus N produces the output $(c + f) \bmod N$, where c is the

correct ciphertext, and f is a function of m, e, N and the offset O caused by the error. f represents the accumulated effect of the error on the encryption.

For Case 1, without loss of generality, we assume that the error occurs during the computation of $c_1 = (m_1)^e \bmod N$ and results in $c_{1\_f} = c_1 + f$, where $0 < f < N$. Hence,

$$(c_{1\_f} \cdot c_2) \bmod N = ((c_1 + f) \cdot c_2) \bmod N = ((c_1 \cdot c_2) \bmod N) + ((f \cdot c_2) \bmod N)$$

$$c_3 = (m_1 \cdot m_2 \bmod N)^e \bmod N = (m_1^e \cdot m_2^e) \bmod N = (c_1 \cdot c_2) \bmod N$$

The proposed CED scheme checks if $c_3 \equiv c_{1\_f} \cdot c_2 \bmod N$ is true, and the error will be missed if and only if $f \cdot c_2 = 0 \bmod N$. This condition is satisfied if and only if ($f = 0 \bmod p$ and $c_2 = 0 \bmod q$) or ($f = 0 \bmod q$ and $c_2 = 0 \bmod p$), where p and q are the prime factors of N, i.e., $N = p \cdot q$. Finding such a combination is essentially as hard as factoring the modulus N.

For Case 2, the CED scheme checks if $c_3 + f \equiv c_1 \cdot c_2 \bmod N$ is true when an error occurs during the computation of $c_3 = (m_1 \cdot m_2 \bmod N)^e \bmod N$ or checks if $c_3 \equiv c_1 \cdot c_2 + f \bmod N$ is true when an error occurs during the computation of $(c_1 \cdot c_2) \bmod N$. In either case, the equation is true only when $f = 0 \bmod N$. The error will never be missed since $0 < f < N$ and f cannot be 0 or multiples of N.

Hence, Lemma 5.3.1 is proved.

Similarly, it can be proved that in the modified CED and the enhanced CED scheme missing a fault is as hard as factoring the modulus N.

5.3.2 Accidental Faults

As analyzed in §5.3.1, for Case 1 when an error occurs during the encryption of m_1, the error will be missed only when ($f = 0 \bmod p$ and $c_2 = 0 \bmod q$) or ($f = 0 \bmod q$ and $c_2 = 0 \bmod p$). Assume that the values of f and c_2 are distributed uniformly in $[1, N-1]$. The probability of f or c_2 being a multiple of p (i.e., $f = 0 \bmod p$ or $c_2 = 0 \bmod p$) is $\frac{q-1}{N-1}$. Similarly, the probability of f or c_2 being a multiple of q is $\frac{q-1}{N-1}$. Hence, the probability of missing an error is $\frac{2\cdot(p-1)\cdot(q-1)}{(N-1)^2} = O(\frac{1}{N})$ where N is of length at least 1024 bits. The same probability can be obtained when an error occurs during the encryption of m_2.

While a transient fault affects one operation, a permanent fault may affect more than one operation. Without loss of generality, we add an offset to the result of any of the four computations and the scheme checks if $(c_1 + f_1) \cdot (c_2 + f_2) \equiv (c_1 \cdot c_2 + f_3) \bmod N$ is true where $f_1, f_2, f_3 \in [0, N-1]$ and at least one of f_1, f_2, f_3 is nonzero. We can prove that the upper bound of the probability of missing errors is $O(\frac{1}{N})$. As the length of N increases, the probability of approaches a negligible value fast.

5.4 Performance and Cost

In this section, we compare the proposed CED scheme to the straightforward CRT-RSA, Giraud scheme (29) and Vigilant scheme (28) in terms of the time overhead, output latency, fault detection latency and hardware overhead. Giraud scheme and Vigilant scheme are two typical effective countermeasures against fault attacks. Since they are given for signature generation, we make our comparison in the signature mode. Table XIII shows the major steps of the

schemes for execution time estimation. The time is consumed mainly by computing modular exponentiation, and the time cost by other computations is negligible.

TABLE XIII

MAJOR STEPS OF CRT-RSA, VILIGANT ALGORITHM AND GIGRAUD ALGORITHM

	CRT-RSA	Viligant Scheme	Giraud Scheme
Step 1	$S_p = m^{d \bmod (p-1)} \bmod p$	$m_p = m \bmod p \cdot r^2$	$(S'_p, S_p) \leftarrow$ modified MLE algorithm
Step 2	$S_q = m^{d \bmod (q-1)} \bmod q$	$m'_p = a_p m_p +$ $b_p(1+r) \bmod p \cdot r^2$ Check m'_p	$(S'_p, S_p) \leftarrow$ modified MLE algorithm
Step 3	$S = CRT(S_p, S_1)$	$S_{pr} = (m'_p)^{d'_p} \bmod p \cdot r^2$ Check S_{pr}	$S' = CRT_{blinded}(S'_p, S'_q)$
Step 4		$S'_p = S_{pr} - b_p(1 + d'_p - R_3)$	$S' = CRT_{blinded}(S_p, S_q)$
Step 5		$m_q = m \bmod q \cdot r^2$	$S' \leftarrow m \cdot S' \pmod{p \cdot q}$ Check S'
Step 6		$m'_q = a_q m_q +$ $b_p(1+r) \bmod p \cdot r^2$ Check m'_q	
Step 7		$S_{pr} = (m'_q)^{d'_q} \bmod q \cdot r^2$ Check S_{qr}	
Step 8		$S'_q = S_{qr} - b_q(1 + d'_q - R_4)$	
Step 9		$S = CRT(S'_p, S'_q)$ Check S	

Vigilant scheme extends the modulus by r^2 where r is a 32-bit random number. Besides r, the scheme requires four 64-bit random integers R_1, R_2, R_3 and R_4. It computes $\hat{m_p} = a_p m_p + b_p \cdot (1 + r) \bmod pr^2$, where $b_p = p \cdot (p^{-1} \bmod r^2)$, $a_p = 1 - b_p$. The correctness of S_{pr} can be verified by checking if $b_p S_{pr} \equiv b_p(1 + d'_p r) \bmod pr^2$ and $d'_p \equiv d_p \bmod (p-1)$ hold (i.e., the step 3 of Vigilant scheme). Similarly, the correctness of S_{qr} is verified in the step 7. The scheme also protects the CRT process by checking S in the step 9.

Giraud scheme is based on the modified Montgomery ladder exponentiation algorithm. It computes $R[0]$ and $R[1]$ in each iteration. After each iteration, $R[0]$ and $R[1]$ satisfy $m \cdot R[0] = R[1]$. At the end of the last iteration, $R[0] = m^d$ and $R[1] = m^{d+1}$, and Giraud scheme checks whether $m \cdot R[0] = R[1]$ holds to detect errors.

5.4.1 Performance

In order to compare the time overhead, we define the time consumed by a modular-exponentiation with 32-bit base, 512-bit exponent, and 512-bit modulus as one time unit, T_0. The length of base is chosen to reflect the typical word size and the length of exponent and modulus are selected based on the recommended minimum key length for RSA. To estimate the execution time, we assume that modulus extension will increase the execution time to $(1 + l_{ex}/l)^2 t$, where l is the length of the original modulus, l_{ex} is the length of the modulus after extension and t is the execution time without modulus extension. We also assume the time cost by computing $m^{d+d_{ex}} \bmod p$ is $(1 + d_{ex}/d)t$, where t is the execution time of computing $m^d \bmod p$.

In the comparison, we only take into account the dominating factor, i.e., the modular exponentiations. The results are shown in Table XIV.

TABLE XIV

PERFORMANCE COMPARISON OF CRT-RSA, VIGILANT ALGORITHM AND GIRAUD ALGORITHM AND OUR CED SCHEME

Steps	CRT-RSA	Vigilant	Giraud	The proposed CED schemes		
				$k = 1$ (modified)	$k = 2$ (basic)	$k > 2$ (enhanced)
1	T_0	T_{v1}	$1.50T_0$	see §5.2.2	see §5.2.1	see §5.2.3
2	T_0	T_{v2}	$1.50T_0$			
3	T_{CRT}	$1.4283T_0$	T_{g1}			
4	-	T_{v3}	T_{g2}			
5	-	T_{v1}	T_{g3}			
6	-	T_{v2}	-			
7	-	$1.4283T_0$	-			
8	-	T_{v3}	-			
9	-	T_{v4}	-			
Total Time	$2T_0$	$2.8476T_0$	$3.0T_0$			
Time Overhead	0	42.38%	50.00%	100% or 200%	50%	$1/k$
Worst-case Latency per Encryption	$2T_0$	$2.8476T_0$	$3.0T_0$	$4T_0$	$6T_0$	$2(k+1)T_0$
Worst-case Fault-detection Latency	$2T_0$	$2.8476T_0$	$3.0T_0$	$4T_0$	$6T_0$	$2(k+1)T_0$

CRT-RSA performs two modular exponentiations, i.e., S_p and S_q, and the time consumed by each modular exponentiation is T_0. In the step 3, S is computed by CRT combination which is given in Algorithm 7. The time cost by CRT combination, denoted by T_{CRT}, is negligible compared to the time cost by modular exponentiations (i.e., $T_{CRT} \ll T_0$). Hence, the total execution time is approximately $2T_0$. The output latency of an encryption in the worst case equals the execution time $2T_0$. The fault detection latency in the worst case is also $2T_0$.

In Vigilant scheme, r is a 32-bit random number. In the step 3 and step 7, it performs two modular exponentiations, i.e., $S_{pr} = (m_p)^{d_p} \bmod p \cdot r^2$ and $S_{qr} = (m_q)^{d_q} \bmod q \cdot r^2$, where the length of d_p and d_q is $512 + 64 = 576$ bits. Therefore, the time cost by each modular exponentiation is $(1 + 64/512) \times (1 + (32 \times 2)/512)2 \times T_0 = 1.4238T_0$. The time consumed by other steps is negligible, so the total time cost by Vigilant scheme is about $2.8476T_0$. The overhead is $(2.8476T_0 - 2T_0)/(2T_0) \approx 42.38\%$. The latency of each encryption in the worst case is $12.8476T_0$. The worst-case fault detection latency is also $2.8476T_0$.

Giraud scheme uses the modified Montgomery ladder exponentiation algorithm to compute modular exponentiations. The length of the modulus is $512 + 32 = 544$ bits. Compared to the binary exponentiation algorithm (Algorithm 4), the Montgomery ladder exponentiation algorithm costs 33% more time in average. Hence, it takes $1.33 \times (1 + 32/512)2T_0 \approx 1.5T_0$ to compute each modular exponentiation, and the time cost by the step 3, 4, 5 is negligible. The execution time is approximately $1.5T_0 + 1.5T_0 = 3T_0$, resulting in an overhead of 50%. The output latency and the fault detection latency in the worst case are both $3T_0$.

When $k = 1$ or $k = 2$, the proposed CED scheme does not perform better than Vigilant scheme and Giraud scheme. Increasing k, the time overhead can be reduced to $\frac{1}{k}$. Meanwhile, the worst-case output latency and fault detection latency are increased. Hence, choosing the value of k is a trade off between time overhead and output latency.

5.4.2 Cost

The hardware overhead mainly depends on the length of the modulus and the memory required to store intermediate results. In order to compare the hardware overhead, we define a hardware unit H to represent the hardware for computing a modular exponentiation with 32-bit base, 512-bit exponent, and 512-bit modulus. For example, the hardware of modular exponentiation $S_p = m^{d \pmod{p-1}} \bmod p$ is H, where the the base is m (32-bit), the exponent is $d \pmod{p-1}$ (512-bit), and the modulus is p (512-bit), the factor of modulus N. Note that the hardware for computing $S_p = m^{d \pmod{p-1}} \bmod p$ increases when the length of p is increased. Assume that there is only one module to compute modular exponentiation in the implementation, and all modular exponentiations in the schemes are computed sequentially in the same datapath. The hardware overhead is shown in Table XV.

TABLE XV

COMPARISON OF THE HARDWARE OVERHEAD OF MODULUS EXTENSION

	CRT-RSA	Vigilant	Giraud	The proposed CED Schemes
Total hardware	H	$1.156H$	$1.0625H$	H
Hardware Overhead	0	15.6%	6.25%	0

In CRT-RSA, the length of the modulus is 512-bit, it can be implemented with H hardware. In Vigilant scheme, the largest modulus used is pr^2 (or qr^2), where r is 32-bit. Therefore, the hardware needed by Vigilant scheme is $(512 + 2 \times 32)/512 \times H \approx 1.156H$, and the hardware overhead is around 15.6%. Giraud scheme uses 544-bit modulus to compute S'_p, S_p, S'_q, and S_q in modified Montgomery ladder exponentiation algorithm. The hardware overhead caused by modulus extension is around 6.25%. The proposed CED scheme does not extend modulus, and hence needs no additional hardware.

The memories required by CRT-RSA, Vigilant scheme, Giraud scheme and the proposed CED scheme for RSA-1024 are given in Table XVI. It is assumed that all algorithms reserve dedicated memory space for the message to be encrypted. Hence the comparison ignores this part.

TABLE XVI

COMPARISON OF THE REQUIRED MEMORY

	CRT-RSA	Vigilant	Giraud	Our Schemes
Memory (bits)	1.5K	3.0K	3.0K	(k+1)K

The implementation of CRT-RSA has to buffer S_p, S_q, and i_q. Each of them is of length 512 bits. Therefore, the memory required by CRT-RSA is $3 \times 512 = 1.5K$ bits. In Vigilant scheme,

the memory consumption peaks right before CRT combination, and the memory required is around $3.0K$ bits. Giraud scheme needs 6×512 bits memory to store S_p, S'_p, S_q, S'_q, S and S'. The proposed CED scheme does not change the implementation of CRT-RSA but needs to buffer the k ciphertexts and the running product of messages (then the ciphtertext of product of the messages). The total memory required is $(k+1)K$ bits.

CHAPTER 6

TOWARDS A COMPREHENSIVE COUNTERMEASURE AGAINST MULTIPLE SIDE CHANNEL ATTACKS

In order to protect public key cryptosystems, researchers have proposed various countermeasures, as discuss in Chapter 3. However, most of them target at preventing only single types of side channel attacks. In practice, however, an attacker can exploit any kind of side channel leakage to attack the cryptosystem. Vulnerability to any side channel attack will compromise the security of the cryptosystem. Hence, a comprehensive protection scheme is expected to thwart multiple side channel attacks.

6.1 Overview

Aware of the threat from multiple side channel attacks, researchers have proposed new schemes to thwart more than one type of side channel attacks. Giraud proposed an approach that can prevent SPA attacks and fault attacks (29). The approach is based on the modified Montgomery ladder exponentiation algorithm (Algorithm 6) which is resistant to SPA. It computes $R[0]$ and $R[1]$ which satisfy $m \cdot R[0] = R[1]$ after each iteration and checks if the relation holds at the end of the last iteration to detect errors. Fumaroli and Vigilant improved Girauds approach by making it thwart DPA attacks (60). While keeping the basic method presented by Giraud to thwart SPA attacks and fault attacks, they use a secret random number to mask the message to break the correlation between power consumption and intermediate data. Kim

and Quisquater showed that the improved approach proposed by Fumaroli and Vigilant caused new leakage which can be expoited to attack it, and they presented an improved one (14). The scheme is based on the infective computation method in (31) rather than error detection method. But the scheme was showed practically insecure in (61). The Bellcore attack (62) is possible when a fault is induced to disturb the infective procedure. Fournaris and Koufopavlou presented a countermeasure (63) which adds a mask and removes it in a similar way to Fumaroli and Vigilant's scheme, and hence the countermeasure suffers the same vulnerability as Fumaroli and Vigilant's scheme.

Moreover, computations are customized in most of the existing countermeasures to thwart certain types of side channel attacks. An disadvantage of such schemes is that they are inflexible. It can be difficult to make them thwart new side channel attacks. Careless modification can cause new leakage compromising the expected security, as shown by Kim and Quisquater (14). Designing a new scheme from scratch whenever resistance to new side channel attacks has be to integrated is unrealistic.

Since there is a rich variety of single-purpose countermeasures which are designed to thwart single types of side channel attacks, it is attractive to design a flexible comprehensive scheme using them. Resistance to new side channel attacks can be easily integrated to the scheme. With this motivation, we discuss the basic principles to construct comprehensive schemes by integrating single-purpose countermeasures without compromising security. Followed the principles, we construct a comprehensive scheme for RSA and then show how to improve it by hardware architecture design. The architecture can enhance the resistance to DPA attacks by

hiding the power consumption of the two exponentiation executions in each other and improve the performance of the implementation.

6.2 Basic Principles to Construct a Flexible Comprehensive Scheme

There are various single-purpose countermeasures. But simply combining them may lead to new leakage enabling new attacks. In this section, we discuss the basic principles to construct a comprehensive scheme using the single-purpose countermeasures without compromising security. In order to keep the effectiveness of the single-purpose countermeasures, the countermeasures integrated in one comprehensive scheme should not affect one another. Following this principle, if the single-purpose countermeasures act on different levels of the design hierarchy, they can be combined directly to a comprehensive scheme without interference. For example, a comprehensive scheme can use Giraud's algorithm-level error detection method (29) to thwart fault attacks while using the dual-rail pre-charge logic in hardware implementation to thwart DPA attacks. However, if the single-purpose countermeasures act on the same level, e.g., the algorithm level, they can be combined safely in a comprehensive scheme if they can be arranged in a nested way shown in Figure 7. The countermeasure of function scope included by another is used as a non-modifiable safe block by the countermeasure of function scope covering it. As a non-modifiable safe block, any modification to its internal computation flow is prohibitive. In Figure 7, Countermeasure 1 is used as a non-modifiable safe block by Countermeasure 2, and similarly, Countermeasure 2 by Countermeasure 3.

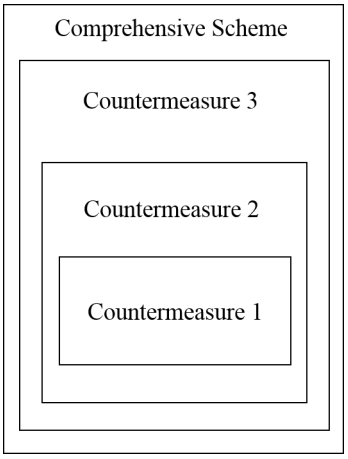

Figure 7. The nested structure of single-purpose countermeasures in a comprehensive scheme

6.3 A Comprehensive Protection Scheme for RSA

Based on the principles proposed in §6.2, we construct a comprehensive protection scheme for RSA against fault attacks and power analysis attacks using the single-purpose countermeasures.

The scheme is shown in Algorithm 14. It uses random numbers to mask the exponent and the message to thwart DPA attacks. The scope of the mask is the entire scheme. The modified Montgomery ladder exponentiation algorithm (Modified MLE) given in Algorithm 15, which can prevent SPA attacks, is used as a non-modifiable safe block to compute modular exponentiations. The two results of the Modified MLE algorithm satisfy a certain relation which can be checked to detect errors. Therefore, the Modified MLE algorithm followed by a verification can thwart fault attacks. The verification process does not affect the mask.

The nested structure of single-purpose countermeasures in the comprehensive scheme is given in Figure 8.

Algorithm 14 The comprehensive scheme for RSA

Input: m, d, N
Output: $s = m^d \bmod N$

Choose random numbers r, k
$d_1 \leftarrow d + k \cdot \varphi(N)$
$(s_{0\_rm}, s_{1\_rm}) \leftarrow$ Modified MLE$(r \cdot m, d_1, N)$
$(s_{0\_r^{-1}}, s_{1\_r^{-1}}) \leftarrow$ Modified MLE(r^{-1}, d_1, N)
$s \leftarrow s_{0\_rm} \cdot s_{0\_r^{-1}} \bmod N$
if $d_1 \equiv d \bmod \varphi(N)$ and $m \cdot s \equiv s_{1\_rm} \cdot s_{1\_r^{-1}} \bmod N$ **then**
    Output s
else
    Supress s, signal error
end if

Algorithm 15 Modified Montgomery ladder exponentiation algorithm (Modified MLE)

Input: $m, d = (d_{l-1}, \ldots, d_1, d_0)_2, N$
Output: $m^d \bmod N$, $m^{d+1} \bmod N$

$R[0] \leftarrow 1$, $R[1] \leftarrow m$
for $i = l - 1$ down to 0 **do**
    $R[\overline{d_i}] \leftarrow R[\overline{d_i}] \cdot R[d_i] \bmod N$
    $R[d_i] \leftarrow R[d_i] \cdot R[d_i] \bmod N$
end for
Return ($R[0]$, $R[1]$)

Now we explain why the scheme works. The inputs to the algorithm are the message to be exponentiated, the exponent d, and the modulus N. It is assumed that d is securely preloaded into the device prior to the RSA process. At the beginning of the algorithm, two random numbers r and k are generated. The secret exponent is then masked by adding a random

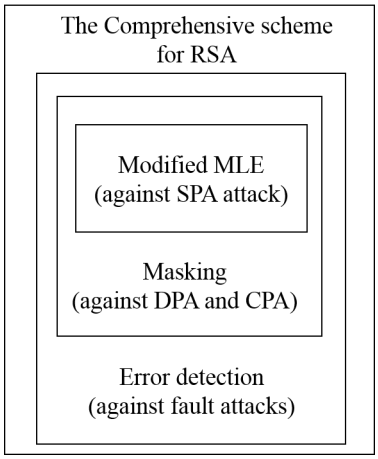

Figure 8. The nested structure of the comprehensive scheme

multiple of $\varphi(N)$ (i.e., $k \cdot \varphi(N)$), where $\varphi(N)$ is Euler's totient function $\varphi(N) = (p-1)(q-1)$. Then the Modified MLE algorithm is performed to compute two modular exponentiations. The Modified MLE algorithm is given in Algorithm 15. It is the same as the Montgomery ladder exponentiation algorithm (Algorithm 3) except outputting both $R[0]$ and $R[1]$. One modular exponentiation takes as inputs the masked exponent d_1, the masked message $r \cdot m$ and the modulus N, and the results are stored in The outputs are stored in s_{0_rm} and s_{1_rm}, respectively. The other modular exponentiation takes r^{-1}, d_1 and N as inputs, where $r \cdot r^{-1} = 1 \bmod N$. The results are stored in $s_{0\_r^{-1}}$ and $s_{1\_r^{-1}}$. Then s is computed by multiplying s_{0_rm} and $s_{0\_r^{-1}}$. Error is detected by checking if $d_1 \equiv d \mod \varphi(N)$ and $m \cdot s \equiv s_{1\_rm} \cdot s_{1\_r^{-1}} \bmod N$ are true.

Lemma 6.3.1. *The comprehensive scheme for RSA outputs the correct result (i.e., $m^d \bmod N$) if no error occurs.*

Proof. The value of s is correct, i.e., $s = m^d \bmod N$ when no error occurs, because

$$\begin{aligned} s = s_{0\_rm} \cdot s_{0\_r^{-1}} \bmod N &= (r \cdot m)^{d_1} \cdot (r^{-1})^{d_1} \bmod N = r^{d_1} \cdot (r^{-1})^{d_1} \cdot m^{d_1} \bmod N \\ &= m^{d_1} \bmod N = m^{d+k\cdot\varphi(N)} \bmod N = m^d \bmod N \end{aligned} \tag{6.1}$$

Next we show that if no error occurs, the verification of $d_1 \equiv d \mod \varphi(N)$ and $m \cdot s \equiv s_{1\_rm} \cdot s_{1\_r^{-1}} \bmod N$ can always succeed. Since $d_1 = d + k \cdot \varphi(N)$, we have $d_1 \equiv d \mod \varphi(N)$. In the Modified MLE algorithm, $R[0]$ and $R[1]$ satisfy $m \cdot R[0] = R[1]$. Hence we have,

$$\begin{aligned} (r \cdot m) \cdot s_{0\_rm} &= s_{1\_rm} \\ (r^{-1}) \cdot s_{0\_r^{-1}} &= s_{1\_r^{-1}} \end{aligned} \tag{6.2}$$

and therefore,

$$\begin{aligned} m \cdot s \bmod N &= m \cdot s_{0\_r^{-1}} \cdot s_{0\_r^{-1}} \bmod N \\ &= m^{d+1} \bmod N \\ &= (r \cdot m)^{d+1} \cdot (r^{-1})^{d+1} \bmod N \\ &= s_{1\_r\cdot m} \cdot s_{1\_r^{-1}} \bmod N \end{aligned} \tag{6.3}$$

□

6.4 Techniques to Improve Resistance and Performance

In this section, we will discuss the techniques to enhance the algorithm-level countermeasure against power analysis attacks and the techniques to improve the performance of the implementation.

6.4.1 Register Transfer Level Techniques

There has been various implementations for RSA to improve performance, reduce hardware overhead and/or lower power consumption (64) (65) (66) (67) (68), but no one exploits the hardware architecture to thwart side channel attacks. The countermeasures against side channel attacks mainly rely on algorithmic techniques. In contrast, we focus the implementation of RSA at Register Transfer Level and takes advantage of techniques at this level to enhance the resistance to power analysis attacks. The basic idea is to implement the modular multiplier based on the pipeline systolic array architecture but schedule the working cells to interleavingly compute two modular multiplications to hide the power consumption. Unlike having two multipliers compute in parallel, it hides power consumption at the cell level. Each cell only consists of several gates.

Modular exponentiation is computed by repeatedly performing modular multiplications and modular squaring operations. Modular squaring operation is nothing but modular multiplication of two identical numbers. Hence the efficiency of modular multiplication will dominate the performance of modular exponentiation. A modular multiplication $x \cdot y \bmod N$ can be computed straightforwardly in two steps: multiplying x and y first and then reducing the product by division-based modulation with modulus N. Since x and y are of the same length as N, their product is twice the length of N. The division-based modulation is expensive in terms of computation and resource. To avoid it, the Montgomery Modular Multiplication (MMM) algorithm performs modular reduction as multiplying progresses (69). An improved version of the MMM algorithm (70), which avoids the final subtraction operation, is shown in Algorithm 16.

In the algorithm, the length of the intermediate results during the computation will not be greater than the length of N. It is much more efficient than the straightforward multiplication and hence is often used to implement high performance modular multiplier.

Algorithm 16 Montgomery Modular Multiplication (MMM) algorithm

Input: $x = (x_l, \ldots, x_1, x_0)_2$, $y = (y_l, \ldots, y_1, y_0)_2$, $N = (n_l, \ldots, n_1, n_0)_2$
$R = 2^{l+2}$, $gcd(N, 2) = 1$
Output: $x \cdot y \cdot R^{-1} \bmod N$

$T \leftarrow 0$
for $i = 0$ to $l + 1$ **do**
    $a \leftarrow (t_0 + x_i \cdot y_0) \bmod 2$
    $T \leftarrow (T + x_i \cdot y + a \cdot N)/2$
end for
return (t)

Note that the MMM algorithm does not compute $x \cdot y \bmod N$ directly. Rather it computes $x \cdot y \cdot R^{-1} \bmod N$ where R could be 2^l where l is the length of x (or y). To compute modular exponentiation using the MMM algorithm, conversions are needed before and after the computation of modular exponentiation. More specifically, as shown in Algorithm 17, $R[0]$ and $R[1]$ will be initialized to R and $m \cdot R$, respectively, before the modular exponentiation starts. And after all the iterations of the *for* loop, $R[0] = m^d R$, and $R[1] = m^{d+1} R$. To remove R from the results, each is multiplied by 1 using the MMM algorithm.

Algorithm 17 Modified MLE algorithm using MMM

Input: $m, d = (d_{l-1}, \ldots, d_1, d_0)_2, N$
Output: $m^d \bmod N$
$R[0] \leftarrow R$, $R[1] \leftarrow MMM(m, R^2, N)$
for $i = l - 1$ down to 0 **do**
    $R[\overline{d_i}] \leftarrow MMM(R[\overline{d_i}], R[d_i], N)$
    $R[d_i] \leftarrow MMM(R[d_i], R[d_i], N)$
end for
$R[0] \leftarrow MMM(R[0], 1, N)$
$R[1] \leftarrow MMM(R[1], 1, N)$
output ($R[0]$, $R[1]$)

In Algorithm 16, $T + x_i \cdot y$ is the partial product, and a is the LSB of the partial product. a is used to adjust the value of the partial product to make it divisible by 2. After $l+2$ iterations, it outputs $x \cdot y \cdot R^{-1} \bmod 2N$. Using it in the Modified MLE algorithm will result in all operations performed modulo $2N$, and it has been proved that after multiplying $R[0]$ and $R[1]$ with 1, the results are smaller than N (66).

We use $T^{(i)}$ and $a^{(i)}$ to indicate the value of T and m after the ith iteration in the *for* loop of the MMM algorithm, and $T^{(i)}$ is computed as follows.

$$T^{(i)} = (T^{(i-1)} + x_i \cdot y + a^{(i)} \cdot N)/2 \tag{6.4}$$

where $i = 0, \ldots, l + 1$ and $T^{(-1)} = 0$. And the jth bit of $T^{(i)}$ is written as $t_j^{(i)}$ and $T^{(i)} = (t_l^{(i)} \ldots t_1^{(i)} t_0^{(i)})$.

Since modular multiplications are the major computation of modular exponentiation, the hardware implementation of the MMM algorithm is the key to the performance of RSA. We can implement the modular multiplier based on the systolic array architecture. To reduce the hardware overhead, we can keep only one row of the systolic array and pipeline it. The architecture is shown in Figure 9. It consists of $l+1$ cells. Cell_0 produces a, and other l cells compute T. The cells ($cell_j$, $j = 1, 2, \ldots, l$) each computes one bit of T, with $j = 1$ for the LSB and $j = l$ for the MSB. Since division by 2 is just a 1-bit right shift, dividing $T^{(i-1)} + x_i \cdot y + a^{(i)} \cdot N$ by 2 is equivalent to storing the jth bit of $(T^{(i-1)} + x_i \cdot y + a^{(i)} \cdot N)$ to the $(j-1)$th bit of $T^{(i)}$. Suppose every cell takes one clock cycle to output a bit of T. Computing $T^{(i)}$ will cost $l+1$ clock cycles due to the data dependency. However, a cell does not have to wait for the completion of $T^{(i)}$ to start computing for $T^{(i+1)}$, since a cell only takes one bit of T as input. Cell_j starts computing $t_{j-1}^{(i+1)}$ once $t_j^{(i)}$ is outputted by cell $j+1$.

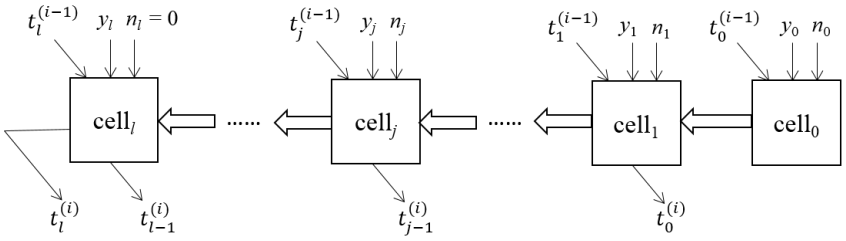

Figure 9. The systolic architecture of modular multiplier

We given an example to show the working state of the cells with $l = 4$ in Figure 10. The number in the cell columns indicate the number of iteration for which the cell is computing in a certain clock cycle. For example, cell_0 computes $a^{(0)}$ in the 1st clock cycle, and the result $a^{(0)}$ is inputted to cell_1 to compute $t_0^{(0)}$ in the 2nd clock cycle. Then cell_2 uses $t_0^{(0)}$ to compute $t_1^{(0)}$ in the 3rd clock cycle. Meanwhile, cell_0 uses $t_0^{(0)}$ to compute $a^{(1)}$. The outputs will be fed into their left cells for computing in the 4th clock cycle and so on. The computation for a certain iteration i moves along the chain of cells from the rightmost cell to the leftmost cell as time elapses.

clock cycle	cell_4	cell_3	cell_2	cell_1	cell_0
1					0
2				0	
3			0		1
4		0		1	
5	0		1		2
6		1		2	
7	1		2		3
8		2		3	
9	2		3		4
10		3		4	
11	3		4		5
12		4		5	
13	4		5		
14		5			
15	5				

Figure 10. An example: the working state of cells in systolic array

From Figure 10, we can see that each cell only computes in alternate clock cycles due to the data dependency. For instance, cell_1 only computes in clock cycle $2, 4, 6, 8, \ldots$. We exploit this property to implement a multiplier that interleaves two modular multiplications. One modular multiplication is computed in the gaps of the other. Therefore, each cell in the systolic array is occupied in all clock cycles, computing for one modular multiplication in odd clock cycles and for the other in even clock cycles. The schedule of cells for interleaved computing is shown in Figure 11. We use Mul_1 and Mul_2 to indicate the two modular multiplications. By interleaving computing, the multiplier takes $3l + 4$ clock cycles to compute two modular multiplications, only one more cycle than performing one modular multiplication. Therefore the time cost by computing the modular exponentiations in the comprehensive scheme (Algorithm 14) is significantly reduced.

The architecture of the interleaving modular multiplier is given in Figure 12. When input signal *start* is set, the controller will enable registers to load input values and set T_1 and T_2 to 0. T_1 and T_2 hold the running products of the two modular multiplications, respectively. The cells each uses either mul_sel or $\overline{mul\_sel}$ to select the inputs for Mul_1 or Mul_2. When the computations of T_1 and T_2 of an iteration completes, registers of x_1 and x_2 are shifted to the right by one bit and the MSB is filled with bit 0. After $3l + 4$ clock cycles, the values stored in T_1 and T_2 are the reduced products of the two modular multiplications.

The implementation of the interleaving modular multiplier is given in Figure 13. It is based on the systolic architecture in Figure 9 with additional logics to interleave two modular multiplications. Cell_0 performs $t_0 + x_i \cdot y_0 \bmod 2$ to produce a, and the other cells compute for

clock cycle	$cell_l$	$cell_{l-1}$	.....	$cell_j$	.....	$cell_3$	$cell_2$	$cell_1$	$cell_0$
1									Mul_1
2								Mul_1	Mul_2
3							Mul_1	Mul_2	Mul_1
4						Mul_1	Mul_2	Mul_1	Mul_2
⋮						⋮	⋮	⋮	⋮
j+1				Mul_1	.....	Mul_2	Mul_1	Mul_2	Mul_1
⋮				⋮		⋮	⋮	⋮	⋮
l		Mul_1	.....	Mul_2	.....	Mul_1	Mul_2	Mul_1	Mul_2
l+1	Mul_1	Mul_2	.....	Mul_1	.....	Mul_2	Mul_1	Mul_2	Mul_1
⋮	⋮	⋮		⋮		⋮	⋮	⋮	⋮
2l+4	Mul_2	Mul_1	.....	Mul_2	.....	Mul_1	Mul_2	Mul_1	Mul_2
2l+5	Mul_1	Mul_2	.....	Mul_1	.....	Mul_2	Mul_1	Mul_2	
⋮	⋮	⋮		⋮		⋮	⋮	⋮	⋮
3l+3	Mul_1	Mul_2							
3l+4	Mul_2								

Figure 11. The schedule of cells for interleaving modular multiplications

$(T + x_i \cdot y + a \cdot N)/2$. Dividing by 2 is just a right shift. Therefore, the cells mainly perform the additions (i.e., $T + x_i \cdot y + a \cdot N$). The cells work like a carry ripple adder. But they "ripple" two carry bits (i.e., $c0$ and $c1$) rather than one in order to add the three vectors. $a^{(i)}$ and x_i are passed through the cell chain since the cells do not synchronize to compute for the same iteration, as we have explained, and they need to know the value of a and x of the iteration they are computing for. Every cell computes for two modular multiplications in alternate clock cycles. This is achieved by multiplexers which select inputs of Mul_1 and Mul_2 alternately and store outputs to proper registers.

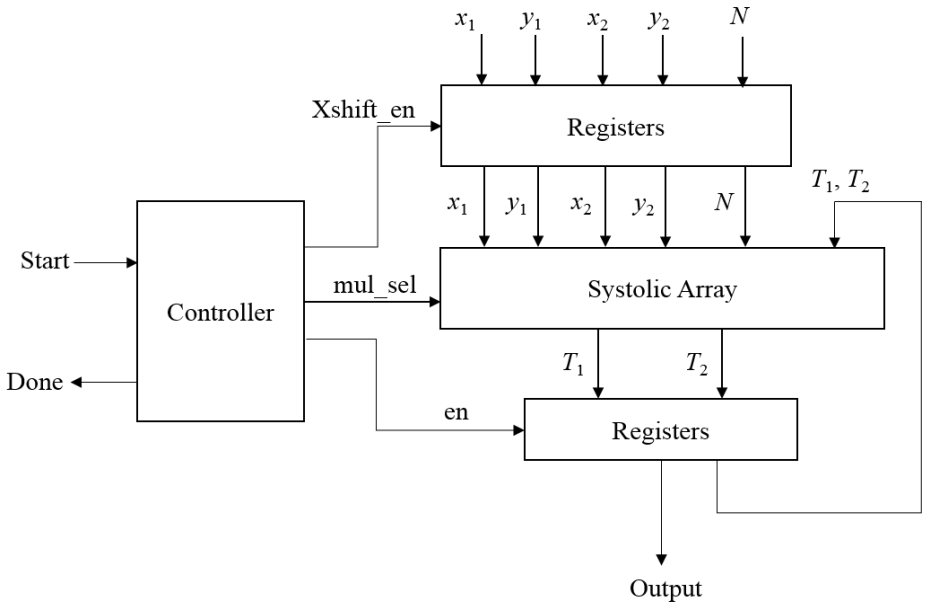

Figure 12. The architecture of the modular multiplier

We denote the carry bits produced by cell_{j-1} for the ith iteration as $c0_{j-1}^{(i)}$ and $c1_{j-1}^{(i)}$. And we still denote T after the ith iteration as $T^{(i)}$ and $T^{(i)} = (t_l^{(i)} \dots t_1^{(i)} t_0^{(i)})$ with $t_j^{(i)}$ indicating the jth bit of $T^{(i)}$. For cell_j $(j \neq 0)$ computing for the ith iteration, it takes as input $t_j^{(i-1)}$, x_i, y_j, $a^{(i)}$, n_j as well as the two carry bits produced by cell_{j-1} (i.e., $c1_{j-1}^{(i)}$ and $c0_{j-1}^{(i)}$) and computes $t_{j-1}^{(i)}$, $c0_j^{(i)}$ and $c1_j^{(i)}$ such that

$$2^2 \times c1 + j^{(i)} + 2 \times c0_j^{(i)} + t_{j-1}^{(i)} = t_j^{(i-1)} + x_i \times y_i + a^{(i)} \times n_j + 2 \times c1_{j-1}^{(i)} + c0_{j-1}^{(i)} \qquad (6.5)$$

The circuit of the cell is shown in Figure 14. It consists of two full adders, one half adder and two AND gates.

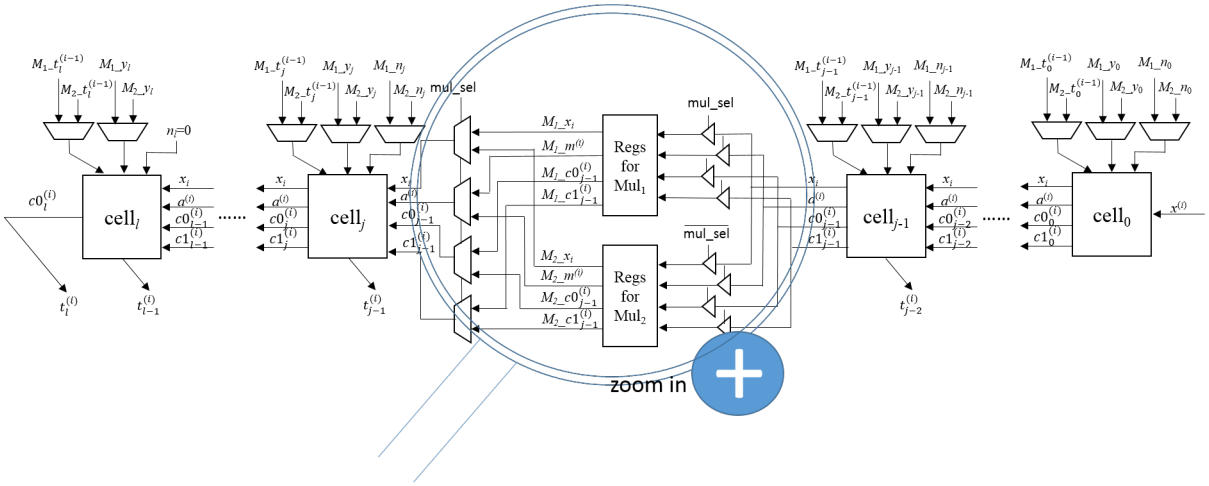

Figure 13. Implementation of the interleaving modular multiplier

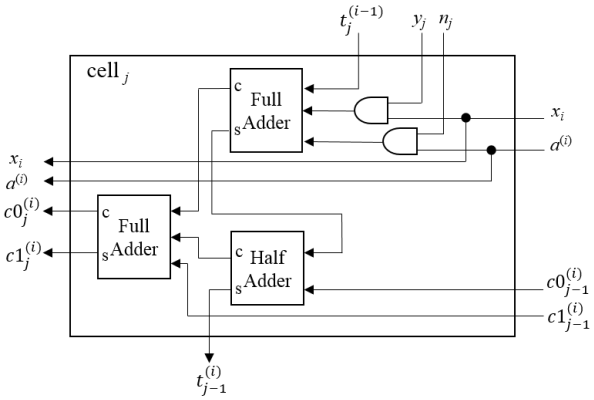

Figure 14. The circuit of a cell the modular multiplier

We use two modular multipliers to implement the comprehensive scheme (Algorithm 14). One multiplier computes modular multiplications and the other computes modular squaring

operations. The hardware architecture is given in Figure 15. Both modular multipliers are based on the systolic architecture in Figure 13 with scheduling method given in Figure 11. Multiplier 1 interleaves the modular multiplications in Modified MLE($r \cdot m$, d_1, N) and Modified MLE($r - 1$, d_1, N), and Multiplier 2 computes the modular squaring operations in the two modular exponentiations.

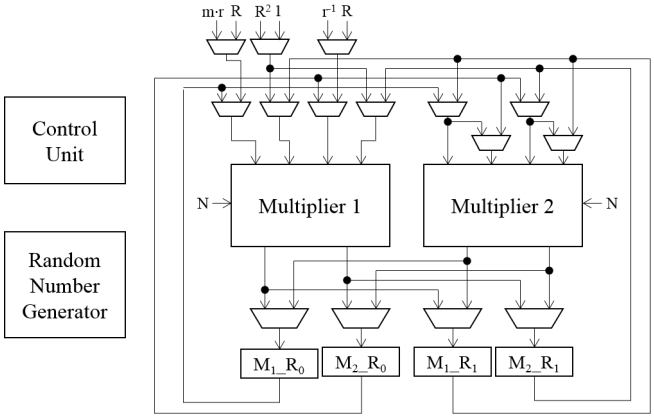

Figure 15. The hardware architecture of the comprehensive scheme

6.5 Mask Reusing

The masks r and k prevent attackers from predicting the intermediate data and hence can protect power analysis attacks. Besides masking the message with r, we also mask the secret exponent using k to thwart the SEMD attack (37). While updating r and k for every message

result in good randomness, it will cause high computational overhead. In this section, we discuss how to reuse the masks to improve the performance.

First of all, k can be reused for a number of modular exponentiations since SEMD attack requires to perform a great number of modular exponentiations with the same exponent to perform the attacks.

Second, we also consider reuse r. Generally speaking, if r is also reused when k is reused, CPA may attack it by inputting messages $X \pmod N$ and $-X \pmod N$ and generating collisions as described by Yen et al. (41). However, with our proposed Register-Transfer level techniques, we show that reusing r is possible. An advantage of the proposed architecture of the comprehensive scheme is that it interleaves two modular exponentiations and hence making their power consumption indistinguishable. Based on this fact, a random number r can be used for two consecutive executions.

The mask reusing scheme is given in Figure 16. Suppose there are n consecutive messages to be exponentiated in the RSA application.

Message flow ⟶		m_1	m_2 m_3	m_4	m_5	m_6 m_7	m_8 ...
Exponentiations	$(r_1^{-1})^{d_1}$	$(r_1 \cdot m_1)^{d_1}$ $(r_2^{-1})^{d_1}$	$(r_1 \cdot m_2)^{d_1}$ $(r_2 \cdot m_3)^{d_1}$	$(r_2 \cdot m_4)^{d_1}$ $(r_3^{-1})^{d_1}$	$(r_3 \cdot m_5)^{d_1}$ $(r_4^{-1})^{d_1}$	$(r_3 \cdot m_6)^{d_1}$ $(r_4 \cdot m_7)^{d_1}$	$(r_4 \cdot m_8)^{d_1}$ $\left(r_5^{-1}\right)^{d_1}$
Output		$(m_1)^d$	$(m_2)^d$, $(m_3)^d$	$(m_4)^d$	$(m_5)^d$	$(m_6)^d$, $(m_7)^d$	$(m_8)^d$

Figure 16. Mask reusing for consecutive messages

Before processing the first message, random numbers r_1 and k_1 are generated and $({r_1}^{-1})^{(d_1)}$ is computed, where $d_1 = d + k_1\varphi(N)$. d is masked by k_1 for the n messages assuming n is far smaller than the number of exponentiations required by DPA attacks. And r_1 is used to mask messages m_1 and m_2. The modular exponentiation of masked m_1, i.e., $(r_1 \cdot m_1)^{d_1}$, is interleaved with the exponentiation of a new random number r_2^{-1}, i.e., $(r_2^{-1})^{d_1}$. Then $(m_1)^{d_1}$ is obtained by multiplying $(r_1^{-1})^{d_1}$ and $(r_1 \cdot m_1)^{d_1}$. r_2 is used to mask messages m_3 and m_4. Modular exponentiations $(r_1 \cdot m_2)^{d_1}$ and $(r_2 \cdot m_3)^{d_1}$ are interleaved. And $(m_2)^{d_1}$ and $(m_3)^{d_1}$ can be obtained by computing $(r_1^{-1})^{d_1} \cdot (r_1 \cdot m_2)^{d_1}$ and $(r_2^{-1})^{d_1} \cdot (r_2 \cdot m_3)^{d_1}$, respectively. Then modular exponentiation of m_4 masked by r_2, i.e., $(r_2 \cdot m_4)^{d_1}$, is interleaved with the modular exponentiation of a new random number r_3^{-1}, i.e., $(r_3^{-1})^{d_1}$. Similar arrangement of computations continues until all the messages have been processed. The arrangement of the modular exponentiations ensures that different random numbers are used in the two modular exponentiations computed interleavingly. It can prevent attackers from predicting the relation between the two modular exponentiations. An attacker cannot generate useful collisions between executions by just choosing inputs without knowing the values of the random numbers.

Mask reusing can reduce computational overhead and hence improve the performance of RSA implementation. If the mask is updated every execution, two modular exponentiations, i.e., $(r \cdot m)^{d_1}$ and $(r^{-1})^{d_1}$ are computed for one message. Compared to the RSA implementation without countermeasures where one modular exponentiation is computed for one message, the computational overhead is around 100%. But if a mask is used for two messages, the computational overhead will be reduced to 50% because $(r^{-1})^{d_1}$ is computed once for two messages.

Comparison of the computational overheads for different updating frequencies of the mask r is summarized in Table XVII.

TABLE XVII

COMPUTATIONAL OVERHEADS OF DIFFERENT MASK UPDATING FREQUENCIES

Mask updating frequency	Number of exponentiations	Overhead
Updating r per message	2	100%
Updating r per two messages	3/2	50%

6.6 Security Analysis

In this section, we evaluate resistance of the proposed comprehensive scheme to fault attacks and power analysis attacks.

6.6.1 Resistance to Fault Attacks

Fault attacks are prevented by error detection. Therefore, we analyze the error detection capability to show that an attacker can hardly induce a fault without being detected.

The proposed scheme computes modular exponentiations using the Modified MLE algorithm. The Modified MLE algorithm outputs $R[0]$ and $R[1]$ which satisfy $m \cdot R[0] = R[1]$. According to Algorithm 14, the two modular exponentiations output $(s_{0\_rm}, s_{1\_rm}) = ((rm)^d, (rm)^{d+1})$ and $(s_{0\_r^{-1}}, s_{1\_r^{-1}}) = ((r^{-1})^d, (r^{-1})^{d+1})$, respectively. As proved in §6.3, if no error occurs, $m \cdot s_{0\_rm} \cdot s_{0\_r^{-1}} \equiv s_{1\_rm} \cdot s_{1\_r^{-1}}$ is true. Error is detected by checking this equivalence. In-

tuitively, if an error occurs during the computation of one modular exponentiation, the expected relation will be broken, resulting in $(r \cdot m) \cdot s_{0\_rm} \neq s_{1\_rm}$ or $r^{-1} \cdot s_{0\_r^{-1}} \neq s_{1\_r^{-1}}$ depending on which modular exponentiation is affected by the error. Consequently, equation $m \cdot s_{0\_rm} \cdot s_{0\_r^{-1}} \equiv s_{1\_rm} \cdot s_{1\_r^{-1}}$ no longer holds. The error can be detected. We will prove that the intuition is true.

Our analysis assumes a single transient fault. If a transient fault lasts within one clock cycle, it may affect only one of the two modular exponentiations. If a transient fault lasts longer than one clock cycle, it may affect both modular exponentiations. This is because the modular multiplications of the two modular exponentiations are interleaved in the proposed architecture.

We begin our analysis with the scenario of a transient fault lasting one clock cycle. According to Algorithm 14, it may effect one of the following computations:

case 1: exponent masking, i.e., $d_1 \leftarrow d + k \cdot \varphi(N)$,

case 2: one of the two modular exponentiations, i.e., Modified MLE$(r \cdot m, d_1, N)$ and Modified MLE(r^{-1}, d_1, N),

case 3: the modular multiplication, i.e., $s \leftarrow s_{0\_rm} \cdot s_{0\_r^{-1}} \bmod N$,

case 4: the verification, i.e., checking if $d_1 \equiv d \bmod \varphi(N)$ and $m \cdot s \equiv s_{1\_rm} \cdot s_{1\_r^{-1}} \bmod N$.

If an error occurs during the verification while the computations for RSA are correct, the verification will result in a false alarm but it does not compromise the security. Hence, we will focus on the error detection capability in case 1, 2 and 3.

In case 1, the error results in $d_1 \neq d + k \cdot \varphi(N)$ and this error can be detected by checking if equation $d_1 \equiv d + k \cdot \varphi(N)$ is true in the verification step.

In case 2, the error occurs during the computation of modular exponentiations.

Lemma 6.6.1. *Inducing a fault during the computation of modular exponentiations such that the error is undetected is as hard as factoring the modulus N.*

Proof. The error can affect either of the two modular exponentiations. We will show that whichever modular exponentiation is affected, the expected equivalence is broken. Without loss of generality, suppose an error occurs in the ith iteration in the *for* loop of the Modified MLE. Before the ith iteration, $R[0]$ and $R[1]$ satisfy $m \cdot R[0]^{(i-1)} = R[1]^{(i-1)}$ where $R[1]^{(i-1)}$ and $R[1]^{(i-1)}$ indicate the value of $R[0]$ and $R[1]$ after the $(i-1)$th iteration, respectively. If $d_i = 0$, the algorithm computes in the ith iteration:

$$\begin{aligned} R[0]^{(i)} &= \left(R[0]^{(i-1)}\right)^2 \\ R[1]^{(i)} &= R[0]^{(i-1)} \cdot R[1]^{(i-1)}. \end{aligned} \tag{6.6}$$

If the error occurs in the modular squaring operation, the erroneous $R[0]$ can be written as

$$R'[0]^{(i)} = \left(R[0]^{(i-1)}\right)^2 + \varepsilon_0 \tag{6.7}$$

where ε_0 is the error function. So after the ith iteration, we have

$$\begin{aligned} m \cdot R'[0]^{(i)} &= m \cdot ((R[0]^{(i-1)})^2 + \varepsilon_0) = m \cdot (R[0]^{(i-1)})^2 + m \cdot \varepsilon_0 \\ R[1]^{(i)} &= R[0]^{(i-1)} \cdot R[1]^{(i-1)} = m \cdot (R[0]^{(i-1)})^2. \end{aligned} \tag{6.8}$$

Hence, $m \cdot R'[0]^{(i)} \neq R[1]^{(i)}$ if $m \cdot \varepsilon_0 \neq 0 \bmod N$. After all iterations, it will cause $(r \cdot m) \cdot s_{0\_rm} \neq s_{1\_rm}$ or $r^{-1} \cdot s_{0\_r^{-1}} \neq s_{1\_r^{-1}}$. Consequently, the error will be detected by coherence checking since $m \cdot s_{0\_rm} \cdot s_{0\_r^{-1}} \neq s_{1\_rm} \cdot s_{1\_r^{-1}}$. If an attacker wants to find the solution to $m \cdot R'[0]^{(i)} = R[1]^{(i)}$ to pass the verification, he/she needs to make $m \cdot \varepsilon_0$ be a multiple of modulus N, i.e., $m \cdot \varepsilon_0 = \alpha \cdot N$, where α is an integer and $\alpha \neq 0$. The modulus N is the product of two large primes p and q, therefore the equation can be written as

$$m \cdot \varepsilon_0 = \alpha \cdot p \cdot q \tag{6.9}$$

Since $m < N$ and $\varepsilon_0 \neq 0 \bmod N$, the solution to the above equation is

$$\begin{cases} m = \alpha_1 p \\ \varepsilon_0 = \alpha_2 q \end{cases} \quad (\alpha_1 \cdot \alpha_2 = \alpha) \tag{6.10}$$

or

$$\begin{cases} m = \alpha_1' p \\ \varepsilon_0 = \alpha_2' q \end{cases} \quad (\alpha_1' \cdot \alpha_2' = \alpha) \tag{6.11}$$

Since the solution requires to know p and q, manipulating a fault such that the error is missed is as hard as factoring the modulus N.

If the error occurs in the modular multiplication, the erroneous $R[1]$ can be written as

$$R'[1]^{(i)} = R[0]^{(i-1)} \cdot R[1]^{(i-1)} + \varepsilon_1 = m \cdot (R[0]^{(i-1)})^2 + \varepsilon_1 \tag{6.12}$$

And $m \cdot R[0]^{(i)} = m \cdot (R[0]^{(i-1)})^2 \neq R'[1]^{(i)}$ since $\varepsilon_1 \neq 0$. The error will be detected by the equivalence verification.

Similarly we can reach the same conclusion in the case of $d_i = 1$. Hence, an error occurring during the computation of modular exponentiations can be detected. Inducing a fault such that the error is missed is as hard as factoring the modulus N. It proves Lemma 6.6.1. □

In case 3, the error occurs during the modular multiplication $s_{0\_rm} \cdot s_{0\_r^{-1}} \bmod N$.

Lemma 6.6.2. *Inducing a fault to the computation of* $s_{0\_rm} \cdot s_{0\_r^{-1}} \bmod N$ *such that the error is undetected is as hard as factoring the modulus* N.

Proof. The error affects the computation of $s_{0\_rm} \cdot s_{0\_r^{-1}} \bmod N$, and the erroneous s can be written as:

$$s' = s_{0\_rm} \cdot s_{0\_r^{-1}} + \varepsilon. \tag{6.13}$$

The verification will not succeed since

$$m \cdot s' = m \cdot s_{0\_rm} \cdot s_{0\_r^{-1}} + m \cdot \varepsilon \tag{6.14}$$

Hence, $m \cdot s' \neq c_{1\_rm} \cdot s_{1\_r^{-1}}$ if $m \cdot \varepsilon \neq 0 \bmod N$. The error will be missed when $m \cdot \varepsilon = \alpha \cdot p \cdot q$. Applying the analysis method used in case 2, we can come to the same conclusion, that is, inducing such a fault is as hard as factoring the modulus N. This proves lemma 6.6.2. □

If a transient fault lasts longer than one clock cycle, it may affect the results of both modular exponentiations. According to above analysis, an attacker can hardly induces a fault without changing the relation of outputs of the Modified MLE. And it is even harder, if not impossible, to induce a long-lasting fault and control the effect of the error to keep the relations of the outputs.

Suppose the erroneous results of the modular exponentiations are (s'_{0_rm}, s'_{1_rm}) and ($s'_{0\_r^{-1}}$, $s'_{1\_r^{-1}}$), respectively. s'_{1_rm} and $s'_{1\_r^{-1}}$ can be written as

$$\begin{aligned} s'_{1\_rm} &= (r \cdot m) \cdot s'_{0\_rm} + \varepsilon_{rm} \\ s'_{1\_r^{-1}} &= r^{-1} \cdot s'_{0\_r^{-1}} + \varepsilon_{r^{-1}} \end{aligned} \tag{6.15}$$

where ε_{rm} and $\varepsilon_{r^{-1}}$ are the offsets caused by the errors, respectively, and $\varepsilon_{rm} \neq 0 \bmod N$, $\varepsilon_{r^{-1}} \neq 0 \bmod N$.

The errors will be missed when

$$\begin{aligned} m \cdot s'_{0\_rm} \cdot s'_{0\_r^{-1}} &= s'_{1\_rm} \cdot s'_{1\_r^{-1}} \bmod N \\ &= (r \cdot m \cdot s'_{0\_rm} + \varepsilon_{rm}) \cdot (r^{-1} \cdot s'_{0\_r^{-1}} + \varepsilon_{r^{-1}}) \bmod N \\ &= s'_{0\_rm} \cdot s'_{0\_r^{-1}} + rm \cdot s'_{0\_rm} \cdot \varepsilon_{r^{-1}} + r^{-1} \cdot s'_{0\_r^{-1}} \cdot \varepsilon_{rm} + \varepsilon_{rm} \cdot \varepsilon_{r^{-1}} \bmod N. \end{aligned} \tag{6.16}$$

Therefore, ε_{rm} and $\varepsilon_{r^{-1}}$ need to satisfy

$$r \cdot m \cdot s'_{0\_rm} \cdot \varepsilon_{r^{-1}} + r^{-1} \cdot s'_{0\_r^{-1}} \cdot \varepsilon_{rm} + \varepsilon_{rm} \cdot \varepsilon_{r^{-1}} = 0 \bmod N. \tag{6.17}$$

Since r is unknown, an attacker can hardly form ε_{rm} and $\varepsilon_{r^{-1}}$ to pass the verification. Moreover, ε_{rm} and $\varepsilon_{r^{-1}}$ are dependent on one another since the location of the fault is fixed once it is induced. It makes fault injection even harder.

6.6.2 Resistance to Power Analysis Attacks

The proposed comprehensive scheme is based on the Montgomery ladder exponentiation algorithm. Hence, it is resistant to SPA attacks.

The scheme is only subject to certain DPA attacks as some attacks assume certain modular exponentiation algorithms other than the Montgomery ladder algorithm. The attack proposed by Walter (71) assumed the exponentiation algorithm whose pattern of operations is dependent on the value of secret exponent and performed power analysis to distinguish modular multiplications and modular squaring operations based on the fact that a conditional subtraction in the MMM algorithm with final subtraction is slightly more likely to occur in a modular squaring operation than in a modular multiplication. The attack presented by Witteman et al. (39) assumed the RSA was computed by the Square-and-multiply-always algorithm, and analyzed the correlation between power consumption of two consecutive modular operations to identify the dummy operation in the algorithm. Such power analysis approaches cannot be used to attack the proposed comprehensive scheme.

For the DPA and CPA attacks which are effective to attack the proposed comprehensive scheme, resistance to them is achieved by the masking at the algorithm level as well as the power consumption hiding at the Register Transfer Level. Masking at the algorithm level randomizes the intermediate data to prevent attackers from predicting it for power analysis. And interleaving computing at the Register-Transfer Level can enhance the resistance to power analysis attacks. We can see from Figure 11 that in any clock cycle except the first and the last one, half cells are computing for Mul_1 and the rest half for Mul_2. As all computing cells contribute to the power consumption of the modular multiplier, the power consumption in any clock cycle is a mixed one of the computations of Mul_1 and Mul_2. The power consumed by computing Mul_1 is comparative to that by computing Mul_2. Therefore, the power consumption of modular multiplications is hided in each other. Consequently, it will be difficult to analyze one of them given the power consumption traces of the modular multiplier.

The three attacks, SEMD, MESD and ZESD presented by Messerges et al. require different capabilities of attackers (37). MESD and ZESD require exponentiating a constant message using exponents chosen by the attacker. Therefore, message masking can prevent the MESD and ZESD attacks. SEMD attack can be prevented by exponent masking since the attack assumes that the attacker is able to ask for exponentiations with a known exponent. The doubling attack (40) and Yen et al.s attack (41) require exponentiations with chosen messages. The messages have to satisfy certain relations (i.e., X and X^2 in doubling attack, and X, $-X$ in Yen et al.'s attack) to generate collisions. The generalized CPA attacks relax the relation to $(m_1)^\alpha = (m_2)^\beta$ (42). The CPA attacks can be prevented by message masking which breaks

the relation between input messages. Exponent masking makes predicting intermediate data even more difficult. The power consumption hiding provided at the Register Transfer Level prevents attackers from exploiting relation between the two modular exponentiations within one execution. The techniques used in the proposed scheme to thwart various power analysis attacks are summarized in Table XVIII.

TABLE XVIII

RESISTANCE TO POWER ANALYSIS ATTACKS

Power Analysis Attacks	Techniques
SPA	Montgomery Ladder Algorithm
Walter's attack (71)	Montgomery Ladder Algorithm
Witteman's attack (39)	Montgomery Ladder Algorithm
MESD, ZESD (37)	Message masking at algorithm level
SEMD (37)	Exponent masking at algorithm level
Doubling attack (40)	Message masking at algorithm level Hiding at Register-Transfer level
Yen et al.'s attack (41)	Message masking at algorithm level Hiding at Register-Transfer level
CPA (42)	Message masking at algorithm level Hiding at Register-Transfer level

While typical DPA attacks analyze the correlation between intermediate data and secret information, Messerges et al. presented a DPA attack which analyzes the correlation between the address values of registers and secret information (72). This attack is also called Address-bit

DPA (ADPA) attack. Itoh et al. (73) and Izumi et al. (74) showed the vulnerability of ECC implemented by the Montgomery ladder algorithm. We did not find related work on RSA, but apparently similar idea can be applied to attack RSA implemented by the Montgomery ladder exponentiation algorithm. Countermeasures against such attacks are based on address-bit randomization when accessing registers. Itoh et al. (73) and Izumi et al. (74) presented countermeasures for ECC. Algorithm 18 shows the same idea applied to RSA.

Algorithm 18 ADPA-resistant Montgomery ladder exponentiation algorithm

Input: m, $d = (d_{l-1}d_{l-2}, \ldots, d_1, d_0)_2$, where $d_{l-1} = 1$, N
Output: $m^d \bmod N$

Generate a random number $r = (r_{l-1} \ldots r_1 r_0)_2$
$R[1 \oplus r_{l-1}] \leftarrow m$, $R[r_{l-1}] \leftarrow m^2$
for $i = l - 2$ downto 0 **do**
    $R[2] \leftarrow (R[d_{i+1} \oplus d_i \oplus r_{i+1}])^2$
    $R[1 \oplus r_i] \leftarrow R[0] \cdot R[1]$
    $R[r_i] \leftarrow R[2]$
end for
output $(R[d_0 \oplus r_0], R[1 \oplus d_0 \oplus r_0])$

6.7 Experiment Results

We model the proposed RSA scheme with $N = 1024$ and $N = 2048$ using VHDL and synthesize it into netlist using Synopsys Design Compiler with TSMC 65 nm library.

6.7.1 Hardware Cost and Performance

The area and performance are given in Table XIX, where T_{RSA}, in the unit of millisecond, indicates the time latency of processing one message.

TABLE XIX

THE AREA AND PERFORMANCE OF THE PROPOSED COMPREHENSIVE SCHEME

N	Area (NAND2-equivalent)	Max f_{clk} (MHz)	T_{RSA} (ms)
1024	584786	289	10.931
2048	1016924	263	47.791

6.7.2 Simulation of Power Analysis Attacks

An IC circuit consumes both leakage power and dynamic power at runtime. From the perspective of power analysis attacks, the dynamic power consumption is the signal and the leakage power consumption is a part of noise. Dynamic power is consumed when a circuit node toggles its state. Hence for a given circuit, the dynamic power consumption of a process is directly related to the toggle activities of all circuit nodes during this process, and the nodal capacitance of each node. Since our implementation does not use bus-like components, we assume most of nodes have roughly the same nodal capacitance. This makes the toggle activities the dominating part of dynamic power consumption.

We want to show that power analysis attacks cannot succeed even if the attackers can make most use of power consumption traces. We remove the negative effect, the noises, caused by measuring device, implementation, etc. We model the power consumption using toggle activities. If power analysis attacks are impossible under this noise-less condition, it is even harder for an attacker to mount the attack in the presence of the negative effect of noises.

We perform gate level simulation of our implementation and then record the toggle activities of all circuit nodes in 3-clock-cycle interval, i.e., about 10 ns. It precisely reflects the transitions in the circuit with zero noise. As we know noise cannot be completely removed in a practical attack, the proposed simulated power consumption is hence the best information an attacker can ever get. Considering the simulation speed, we used $N = 256$ RSA to illustrate the power consumption of proposed scheme. Figure 17 shows a partial power trace with the details. The trace clearly shows the power consumption of a conversion followed by six iterations for computing modular exponentiation. Prior to the iterations for computing modular exponentiation, a modular multiplication is performed to convert initial values by multiplying it with R^2 mod N. Since only one modular multiplier is working during the conversion, the peak power during the conversion is only about half of the peak power of iterations for computing modular exponentiation. The six iterations correspond to a sequence of exponent bit $1, 1, 1, 0, 1, 0$. From the trace, it is hard to tell the value of the exponent bits. The power trace of subsequent iterations continue this uniform pattern and no exponent bits can be derived.

Due to slow simulation speed as well as the large number of sampling points, we are not able to simulate and show the power trace of the entire execution. But we show the power trace of

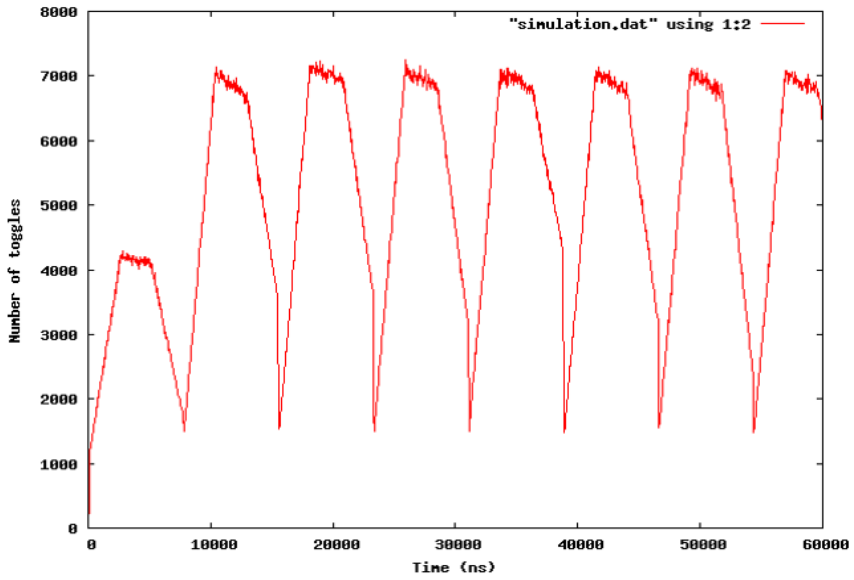

Figure 17. Similation of the power trace using toggle counts

a randomly-picked iterations in the middle of exponentiation in Figure 18. As can be observed, just like the power traces of the initial iterations, the power trace of the middle iterations does not exhibit exploitable pattern to reveal the value of the exponent. Hence it is resistant to SPA.

Resistance to DPA and CPA is gained by masking inputs at the algorithm level and hiding power consumption at the Register Transfer Level. The masks break the correlation between power consumption and intermediate data. Since the message and exponent are masked by random numbers, attacks cannot predict intermediate data to perform statistical analysis to test the hypotheses on the value of key. Hence, message and exponent masking can thwart DPA attacks. Masking can also prevent CPA attacks. Without the knowledge of masks, attackers cannot generate collisions as expected by choosing messages and hence cannot reveal

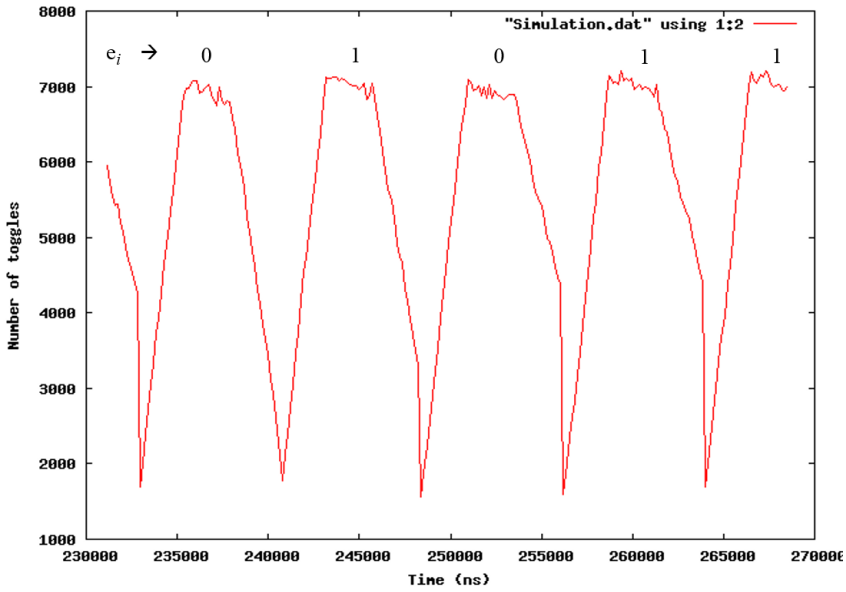

Figure 18. Power trace of middle iterations for computing modular exponentiation

the secret key by comparing the power consumption traces. The implementation, which interleaves two modular multiplications, enhances the resistance to CPA attacks, especially when reusing masks. An attacker cannot extract the power consumption of one modular exponentiation from the mixed power consumption of two. And the scheduling method for mask reusing prevents from predicting the relation between executions to thwart CPA attacks. We simulate the doubling attack, a typical CPA attack and show its power consumption trace of exponent bit sequence $1, 1, 1, 0$ with chose message m and m^2, respectively, in Figure 19.

The value of $R[1]$ after the first iteration shown in the top power trace is equal to the value of $R[1]$ after the iteration right before the first iteration shown in the bottom power trace. When exponent bit is 1, $R[1]$ is used for computing the modular squaring operations

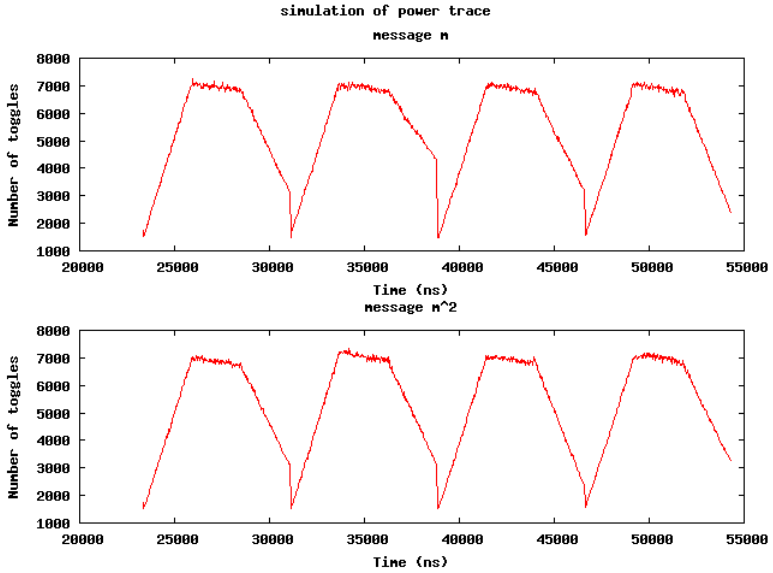

Figure 19. Power consumption of exponentiating m and m^2

in both executions and hence the power consumption waveforms for the two modular squaring operations are supposed to match. Therefore, the value of the exponent bit can be revealed by waveform comparison. We denote the waveforms of four iterations in the top power trace as S_1, S_2, S_3, S_4, respectively, and that in the bottom power trace as W_1, W_2, W_3, W_4, respectively. Based on the value of the exponent bit, W_1 is supposed to match S_2. However, we can see from Table XX that the two waveforms do not match as expected. The waveforms of unrelated iterations have even smaller difference. The reason is that message masking and exponent masking randomize the base as well as the exponent of the modular exponentiation. Consequently, the relation of chosen messages are not reflected on the intermediate data. So doubling attack fails.

TABLE XX

COMPARISON OF THE WAVEFORM

Comparison	W_1/S_1	W_1/S_2	W_1/S_3	W_1/S_4
Difference (average number of toggles)	96	267	76	159

CHAPTER 7

CONCLUSION

In this dissertation, we present new error detection schemes for ECC and RSA to thwart fault attacks. The schemes share the same design idea: exploiting an invariant during the computation and checking the correctness of the invariant to detect errors. The invariants is constructed from the mathematical property. In LOEDAR, the proposed low-cost error detection and recovery scheme for ECC, a new point P_v is introduced into the Montgomery ladder algorithm and updated in a way such that $P_v + P_2 \equiv 2P_1$. The error detection scheme for RSA is based on the homomorphic property.

Both schemes have good error detection capability. The probability of undetected error is quite small and negligible. The good error detection capability enables them to thwart fault attacks. Inducing a fault such that the error is missed by the proposed error detection methods is extremely difficult, if not impossible. Moreover, both schemes can be extended easily to thwart power analysis attacks. The LOEDAR scheme is compatible with most of the existing countermeasures against power analysis attacks. They can be used directly without modifying the error detection method in the LOEDAR scheme to thwart power analysis attacks. The error detection scheme for RSA does not impose any requirement on how to compute the modular exponentiation and hence can work with any RSA implementation. A fine-tuned RSA implementation to thwart other types of side channel attacks, e.g., power analysis attacks, can be used in the error detection scheme.

Experiment results show that the LOEDAR scheme can achieve high performance by properly choosing the verification frequency. The time overhead of the LOEDAR scheme depends on how frequently the EDR process is performed. Each execution of EDR process contributes about 0.68% time overhead. The hardware overhead of the LOEDAR scheme is 37.6%, which is much smaller than other error detection and recovery schemes. The time overhead of the error detection scheme for RSA can be significantly reduced by processing consecutive messages. The larger the number of the consecutive messages, the smaller the time overhead. But the number of consecutive messages processed at a time is limited by output latency of single messages. It should be chosen properly based on the requirement of the RSA application.

We also discuss the basic principles to construct a comprehensive scheme to thwart various side channel attacks. It is important because the security of the public key cryptosystems will be compromised if it can be broken successfully by any side channel attack. Following the principles, a comprehensive scheme can be constructed by integrating the existing single-purpose countermeasures without compromising the security. The constructing method allows countermeasures against new side channel attacks to be integrated into the comprehensive scheme at any time. Hence, it is flexible to extend the comprehensive scheme for new resistance. We present a scheme for RSA to thwart both fault attacks and power analysis attacks. The scheme is constructed by combining the countermeasure against fault attacks and the countermeasure against power analysis attacks. And techniques at the Register Transfer Level are exploited to improve the performance as well as resistance to power analysis attacks. Analysis and experiment results show that the constructed scheme can effectively thwart fault attacks and

power analysis attacks. Based on the constructing principle, the scheme can integrate proper single-purpose countermeasures to thwart other types of side channel attacks.

CITED LITERATURE

1. R. L. Rivest, A. Shamir, and L. Adleman: A Method for Obtaining Digital Signatures and Public-key Cryptosystems. Communications of the ACM 21, pages 120–126, 1978.

2. N. Koblitz: Elliptic Curve Cryptosystems. Mathematics of Computation, 48:203–209, 1987.

3. V. Miller: Use of Elliptic Curves in Cryptography. In CRYPTO 85, pages 417–426, 1985.

4. National Security Agency: NSA Suite B Cryptography, 2009.

5. NIST: Recommendation for Key Management Part 1: General (Revision 3), NIST Special Publication 800-57. Technical report, 2012.

6. W. Diffie and M. E. Hellman: New Directions in Cryptography. IEEE Transactions on Information Theory, IT-22(6):644–654, 1976.

7. RSA Laboratories: PKCS#1. Technical report, 2012.

8. NIST: Digital Signature Standard (DSS), FIPS-186-4. Technical report, 2013.

9. D. Dolev, C. Dwork, and M. Naor: Non-Malleable Cryptography. SIAM Journal on Computing 30(2), pages 391–437, 2000.

10. ANSI Standards Committee X9: Public Key Cryptography for the Financial Services Industry - Key Agreement and Key Transport Using Elliptic Curve Cryptography, ANSI X9.63, 2011.

11. T. Kleinjung, K. Aoki, J. Franke, A. K. Lenstra, E. Thomé, J. W. Bos, P. Gaudry, A. Kruppa, P. L. Montgomery, D. A. Osvik, H. Riele, A. Timofeev, and P. Zimmermann: Factorization of a 768-Bit RSA Modulus. In CRYPTO, pages 333–350, 2010.

12. J. W. Bos, M. E. Kaihara, T. Kleinjung, A. K. Lenstra, and P. L. Montgomery: Solving a 112-bit Prime Elliptic Curve Discrete Logarithm Problem on Game Consoles using Sloppy Reduction. International Journal of Applied Cryptography, 2(3):212–228, 2012.

13. J. Quisquater and D. Samyde: Electromagnetic Analysis (EMA): Measures and Countermeasures for Smart Cards. In Smart Card Programming and Security (E-smart 2001), pages 200–210, 2001.

14. Kim, C. H. and Quisquater, J.-J.: How Can We Overcome Both Side Channel Analysis and Fault Attacks on RSA-CRT? In Workshop on Fault Diagnosis and Tolerance in Cryptography, pages 21–29, 2007.

15. López, J. and Dahab, R.: Improved algorithms for elliptic curve arithmetic in $gf(2^n)$. In Proceedings of the Selected Areas in Cryptography, SAC '98, pages 201–212, London, UK, UK, 1999. Springer-Verlag.

16. P. Montgomery: Speeding the Pollard and elliptic curve methods of factorization. Mathematics of Computations, 48:243–264, 1987.

17. J. Lopez and R. Dahab: Fast multiplication on elliptic curves over GF(2m) without precomputation. In Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems, pages 316–327, 1999.

18. Boneh, D., Demillo, R. A., and Lipton, R. J.: On the Importance of Checking Cryptographic Protocols for Faults. Computations Journal of Cryptology The Journal of the International Association for Cryptologic Research, 1233:101–119, 1997.

19. I. Biehl, B. Meyer, and V. Muller: Differential Fault Attacks on Elliptic Curve Cryptosystems. In Proceedings of Crypto 2000: Advances in Cryptology., pages 131–146. Springer-Verlag, 2000.

20. M. Ciet and M. Joye: Elliptic Curve Cryptosystems in the Presence of Permanent and Transient Faults. Designs Codes and Cryptography, 36(1):33–43, 2005.

21. P.-A. Fouque, R. Lercier, D. Réal, and F. Valette: Fault Attack on Elliptic Curve with Montgomery Ladder Implementation. In Fault Diagnosis and Tolerance in Cryptography (FDTC 08), pages 92–98. IEEE Computer Society, 2008.

22. J. Blömer, Otto, M., and Seifert, J.-P.: Sign Change Fault Attacks on Elliptic Curve Cryptosystems. In Fault Diagnosis and Tolerance in Cryptography 2006 (FDTC 06), volume 4236 of Lecture Notes in Computer Science, pages 36—-52. Prentice Hall, 2006.

23. Yen, S.-M. and Joye, M.: Checking before output may not be enough against fault-based cryptanalysis. IEEE Transactions on Computers, 49(9):967–970, 2000.

24. Coron, J.-S., Joux, A., Kizhvatov, I., Naccache, D., and Paillier, P.: Fault Attacks on RSA Signatures with Partially Unknown Messages. In Proceedings of the 11th International Workshop on Cryptographic Hardware and Embedded Systems, CHES '09, pages 444–456, Berlin, Heidelberg, 2009. Springer-Verlag.

25. A. Domínguez-Oviedo and M. A. Hasan: Error Detection and Fault Tolerance in ECSM using Input Randomization. IEEE Transactions on Dependable and Secure Computing, 6(3):175–187, 2009.

26. Shamir, A.: Improved Method and Apparatus for Protecting Public Key Schemes from Timing and Fault Attacks, 1998.

27. C. Aumuller, P.Bier, W. Fischer, P. Hofreiter, and J.-P. Seifert: Fault Attacks on RSA with CRT: Concrete Results and Practical Countermeasures. In Cryptographic Hardware and Embedded Systems (CHES 2002), pages 260–275, 2002.

28. Vigilant, D.: RSA with CRT: A New Cost-Effective Solution to Thwart Fault Attacks. In CHES '08: Proceeding sof the 10th international workshop on Cryptographic Hardware and Embedded Systems, pages 130–145, Berlin, Heidelberg, 2008. Springer-Verlag.

29. Giraud, C.: An RSA Implementation Resistant to Fault Attacks and to Simple Power Analysis. Computers, IEEE Transactions on, 55(9):1116–1120, 2006.

30. S.-M. Yen, S. Kim, S. Lim, and S. Moon: RSA Speedup with Residue Number System Immune against Hardware Fault Cryptanalysis. In International Conference on Information Security and Cryptology - ICISC 2001, pages 397–413. Springer-Verlag, 2001.

31. J. Blömer, M. Otto, and J.-P. Seifert: A New CRT-RSA Algorithm Secure against Bellcore Attacks. In ACM Conference on Computer and Communications Security, pages 311–320. ACM Press, 2003.

32. S.M. Yen and M. Joye: Checking Before Output may Not Be Enough against Fault-based Cryptoanalysis. IEEE Transactions on Computers, pages 967–970, 2000.

33. Kocher, P., Jaffe, J., and Jun Benjamin: Introduction to Differential Power Analysis and Related Attacks, 1998.

34. Coron, J. S.: Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems. In Cryptographic Hardware and Embedded Systems, eds. c. K. Koç and C. Paar, volume 1717 of Lecture Notes in Computer Science, pages 292–302. Springer, 1999.

35. L. Goubin: A Refined Power-Analysis Attack on Elliptic Curve Cryptosystems. In Public Key Cryptography PKC 2003, ed. Y. Desmedt, volume 2567 of Lecture Notes in Computer Science, pages 199–210. Springer, 2003.

36. T. Akishita and T. Takagi: Zero-Value Point Attacks on Elliptic Curve Cryptosystem. In Information Security ISC 2003, eds. C. Boyd and W. Mao, volume 1 of Lecture Notes in Computer Science, pages 218–233. Springer, 2003.

37. T. S. Messerges, E. A. Dabbish, and R. H. Sloan: Investigations of power analysis attacks on smartcards. In USENIX Workshop on Smartcard Technology, 1999.

38. B. d. Boer, K. Lemke, and G. Wicke: A DPA Attack against the Modular Reduction within a CRT Implementation of RSA. In CHES'02, pages 21–34, 2002.

39. M. F. Witteman, Jasper G. J. van Woudenberg, and F. Menarini: Defeating RSA Multiply-always and Message Blinding Countermeasure. In International Conference on Topics in Cryptology (CT-RSA 2011), pages 77–88, 2011.

40. A.P. Fouque and F.Valette: The Doubling Attack Why Upwards is Better Than Downwards. In Workshop Cryptographic Hardware and Embedded System (CHES03), pages 269–280, 2003.

41. Yen, S.-M., Lien, W.-C., Moon, S., and Ha, J.: Power analysis by exploiting chosen message and internal collisions - vulnerability of checking mechanism for RSA-Decryption. In Proceedings of the 1st international conference on Progress in Cryptology in Malaysia, Mycrypt'05, pages 183–195, Berlin, Heidelberg, 2005. Springer-Verlag.

42. N. Homma, A. Miyamoto, T. Aoki, A. Satoh, and A. Samir: Comparative Power Analysis of Modular Exponentiation Algorithms. IEEE Transactions on Computers, 59(6):795 – 807, 2010.

43. H. Mamiya, A. Miyaji, and H. Morimoto: Efficient Countermeasures against RPA, DPA, and SPA. In Proceedings of Cryptographic Hardware and Embedded Systems CHES, eds. M. Joye and J.-J. Quisquater, volume 3156 of Lecture Notes in Computer Science, pages 343–356. Springer, 2004.

44. C. Kim, J. Ha, S. Moon, S.-M. Yen, W.-C. Lien, and S.-H. Kim: An Improved and Efficient Countermeasure against Power Analysis Attacks, 2005.

45. Y. Wang and L. M. Douglas: A Robust Algorithm for DPA-resistant ECC. In International Symposium on Integrated Circuits, pages 667 – 670, 2009.

46. Kocher, P.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In Advances in CryptologyCRYPTO96, ed. N. Koblitz, volume 1109 of Lecture Notes in Computer Science, pages 104–113. International Association for Cryptologic Research, Springer, 1996.

47. S. N. Cirrus, S. Niranjan, and J. F. Frenzel: A Comparison of Fault-Tolerant State Machine Architectures for Space-Borne Electronics. IEEE Transactions on Reliability, 45:109–113, 1996.

48. M. Ciet and Joye, M.: (Virtually)Free Randomization Techniques for Elliptic Curve Cryptography. In Information and Communication Security (ICICS), pages 348–359, 2003.

49. P. Fouque, D. Real, F. Valette, and M. Drissi: The Carry Leakage on the Randomized Exponent Countermeasure. In Cryptographic Hardware and Embedded Systems (CHES), pages 198–213. LNCS vol. 5154, Springer, 2008.

50. L. Song and K. K. Parhi: Low-Energy Digit-Serial/Parallel Finite Field Multipliers. Journal of VLSI signal processing systems for signal, image and video technology, 19(2):149–166, 1998.

51. S. M. Shohdy, A. El-Sisi, and N. A. Ismail: FPGA Implementation of Elliptic Curve Point Multiplication over GF(2ˆ191). In ISA '09 Proceedings of the 3rd International Conference and Workshops on Advances in Information Security and Assurance, pages 619–634, 2009.

52. M. Morales-Sandoval and C. Feregrino-Uribe: GF(2ˆm) Arithmetic Modules for Elliptic Curve Cryptography. In IEEE International Conference on Reconfigurable Computing and FPGA's, pages 1 – 8, 2006.

53. X. Guo, J. Fan, P. Schaumont, and I. Verbauwhede: Programmable and Parallel ECC Coprocessor Architecture: Tradeoffs between Area, Speed and Security. In Cryptographic Hardware and Embedded Systems (CHES), pages 289–303, 2009.

54. H. R. Zarandi, S. G. Miremadi, and A. Ejlali: Dependability Analysis Using a Fault Injection Tool based on Synthesizability of HDL Models. In Defect and Fault Tolerance in VLSI Systems, pages 485–492, 2003.

55. S. N. Chari, V. V. Diluoffo, P. A. Karger, E. R. Palmer, T. Rabin, J. R. Rao, P. Rohotgi, H. Scherzer, M. Steiner, D. C. Toll, S. Chari, and P. Rohatgi: Designing a Side Channel Resistant Random Number Generator. In 9th IFIP WG 8.8/11.2 international conference on Smart Card Research and Advanced Application, eds. D. Gollmann, J.-L. Lanet, and J. Iguchi-Cartigny, volume 6035 of LNCS, pages 49–64. Springer, 2010.

56. R. Baumann: Soft Errors in Advanced Computer Systems. IEEE Design & Test of Computers, 22:258 – 266, 2005.

57. P. Paillier: Public-key Cryptosystems based on Discrete Logarithms Residues. Eurocrypt, 1592:223–238, 1999.

58. J.C. Lo: A Novel Area-time Efficient Static CMOS Totally Self-checking Comparator. IEEE Journal of Solid-State Circuits, 28:165 –168, 1993.

59. C. Metra, M. Favalli, and B. Ricco: Highly Testable and Compact Single Output Comparator. In Proceedings of 15th IEEE VLSI Test Symposium (VTS), 1997.

60. G. Fumaroli and D. Vigilant: Blinded Fault Resistant Exponentiation. In FDTC06, pages 62–70, 2006.

61. E. Dottax, C. Giraud, M. Rivain, and Y. Sierra: On Second-Order Fault Analysis Resistance for CRT-RSA Implementation. In 3rd IFIP WG 11.2 International Workshop on Information Security Theory and Practice, Smart Devices, Pervasive Systems and Ubiquitous Networks, WISTP09, pages 68–83, 2009.

62. Boneh, D., Demillo, R. A., and Lipton, R. J.: On the importance of eliminating errors in cryptographic computations. Journal of Cryptology, 14:101–119, 2001.

63. Fournaris, A. P. and Koufopavlou, O.: Efficient CRT RSA with SCA Countermeasures. In Proceedings of the 14th Euromicro Conference on Digital System Design, DSD '11, pages 593–599, Washington, DC, USA, 2011. IEEE Computer Society.

64. C. McIvor, M. Mcloone, and J. McCanny: Modified Montgomery Modular Multiplication and RSA Exponentiation Techniques. In IEE Proceedings of Computers and Digital Technques, pages 402–408, 2004.

65. K. Iwamura, T. Matsumoto, and H. Imai: Systolic-arrays for Modular Exponentiation using Montgomery Method. In Advances in Cryptology: Proceedings of EUROCRYPT92, pages 477–481, 1992.

66. S.B. Ors, L. Batina, B. Preneel, and J. Vandewalle: Hardware Implementation of a Montgomery Modular Multiplier in a Systolic Array. In Parallel and Distributed Processing Symposium, 2003. Proceedings. International, 2003.

67. S.-R. Kuang, J.-P. Wang, K.-C. Chang, and H.-W. Hsu: Energy-efficient High-throughput Montgomery Modular Multipliers for RSA Cryptosystem. IEEE Transactions on Very Large Scale Integration(VLSI), 21:1999–2009, 2013.

68. S.S. Ghoreishi, M.A. Pourmina, H. Bozorgi, and M. Dousti: High speed RSA Implementation based on Modified Booths Technique and Montgomerys Multiplication for FPGA Platform. In Advances in Circuits, Electronics and Micro-electronics, pages 86–93, 2009.

69. P. L. Montgomery: Modular Multiplication Without Trial Division. Mathematics of Computation, 44:519–521, 1985.

70. C.D.Walter: Montgomery Exponentiation Needs No Final Subtractions. Electronics Letters, 35(21):1831 – 1832, 1999.

71. C. D. Walter: Longer Randomly Blinded RSA Keys May Be Weaker Than Shorter Ones. In 8th International Workshop on Information Security Applications (WISA'07), pages 303–316, 2007.

72. T. S. Messerges, E. A. Dabbish, and R. H. Sloan: Investigations of Power Analysis Attacks on Smartcards. In USENIX Workshop on SmartCard Technology, 1999.

73. K. Itoh, T. Izu, and M. Takenaka: A Practical Countermeasure against Address-Bit Differential Power Analysis. In Cryptographic Hardware and Embedded Systems CHES 2003, pages 382–396, 2003.

74. M. Izumi, J. Ikegami, K. Sakiyama, and K. Ohta: Improved Countermeasure against Address-bit DPA for ECC Scalar Multiplication. In Design, Automation & Test in Europe Conference (DATE), pages 981–984, 2010.

75. A. Matthews: Side-channel Attacks on Smartcards. Network Security, 2006(12):18–20, 2006.

76. Boscher, A.; Handschuh, H. T. E.: Blinded Fault Resistant Exponentiation Revisited . In Fault Diagnosis and Tolerance in Cryptography (FDTC), 2009 Workshop on, pages 3–9, 2009.

77. D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi: The EM Side-Channel(s). In Cryptographic Hardware and Embedded Systems CHES 2002, eds. B. S. Kaliski JR, c. K. Koç, and C. Paar, volume 2523 of Lecture Notes in Computer Science, pages 29–45. Springer, 2002.

78. D. Boneh: Twenty years of attacks on the RSA cryptosystems. Notices of the American Mathematical Society, 46:203–213, 1999.

79. D. Wagner: Cryptanalysis of a provably secure CRT-RSA algorithm. In CCS '04 Proceedings of the 11th ACM Conference on Computer and Communications Security, pages 92 – 97, 2004.

80. E. Biham and A. Shamir: Differential Fault Analysis of Secret Key Cryptosystems. In Advances in Cryptology Crypto 97, ed. B. Kaliski, volume 1294 of Lecture Notes in Computer Science, pages 513–525. Springer-Verlag, 1997.

81. J. F. Dhem, F. Koeune, P. A. Leroux, P. Mestré, J. Quisquater, and J. L. Willems: A Practical Implementation of the Timing Attack. In Time, ed. J.-J. Quisquater, volume 1820 of Lecture Notes in Computer Science, pages 167–182. Springer, 1998.

82. NIST: The Federal Information Processing Standard (FIPS) Publication 140-3: DRAFT Security Requirements for Cryptographic Modules, 2013.

83. P. Golle, M. Jakobsson, A. Juels, and P. Syverson: Universal Re-encryption for Mixnets. In RSA Conference Cryptographers' Track, pages 163–178, 2004.

84. R. Cramer, I. Damgard, and J. B. Nielsen: Multiparty Computation from Threshold Homomorphic Encryption. In Advances in Cryptography, Eurocrypt, pages 280 – 300, 2001.

85. T. ElGamal: A Public-key Cryptosystem and A Signature Scheme based on Discrete Logarithms. IEEE Transactions on Information Theory, IT-31:469–472, 1985.

86. W. Van Eck: Electromagnetic radiation from video display units: An eavesdropping risk? Computers Security, 4(4):269–286, 1985.

VITA

NAME	Kun Ma
EDUCATION	Ph.D., Electrical and Computer Engineering, University of Illinois at Chicago, Illinois, May 2014
	M.S., Electrical Engineering, Beijing University of Posts and Telecommunications, May 2009
	B.S., Electrical Engineering, Jilin University, July 2006
EXPERIENCE	Research Assistant, Department of ECE, University of Illinois at Chicago, 08/2009 - 05/2014
	Teaching Assistant, Department of ECE, University of Illinois at Chicago, 08/2009 - 12/2013.
PUBLICATIONS	K. Ma, K. Wu, "Error Detection and Recovery for ECC: A New Approach against Side-Channel Attacks", to be published by IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2013.
	X. Cui, K. Ma, K. Wu, High-Level Synthesis for Hardware Trojan Detection and Recovery, Design Automation Conference (DAC), 2014.
	K. Ma, H. Liang, K. Wu, "Homomorphic Property Based Concurrent Error Detection of RSA: A Countermeasure to Fault Attack", IEEE Transactions on Computers, vol. 61, pp. 1040-1049, July 2012.
	K. Ma, K. Wu, "LOEDAR: A Low Cost Error Detection and Recovery Scheme for ECC", in Proceedings of Design, Automation & Test in Europe (DATE), 2011.