國 立 交 通 大 學

電子工程學系 電子研究所碩士班

碩 士 論 文

使用可擴展蒙哥馬利乘法器之抵抗簡單及差動能量攻擊法的RSA 密碼核心

A RSA Crypto-Core using Scalable Montgomery Multiplication with DPA and SPA Resistance

研究生：林祐進

指導教授：張錫嘉 教授

中 華 民 國 九 十 七 年 十一 月

使用可擴展蒙哥馬利乘法器之抵抗簡單及差動能量攻擊法的 RSA 密碼核心

A RSA Crypto-Core using Scalable Montgomery Multiplication with DPA and SPA Resistance

研 究 生：林祐進　　　　Student：Tu-Ching Lin

指導教授：張錫嘉　　　　Advisor：Hsie-Chia Chang

國 立 交 通 大 學

電 子 工 程 學 系　電 子 研 究 所 碩 士 班

碩 士 論 文

A Thesis

Submitted to Department of Electronics Engineering & Institute of Electronics

College of Electrical and Computer Engineering

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of Master

In

Electronics Engineering

November 2008

Hsinchu, Taiwan, Republic of China

中華民國九十七年十一月

使用可擴展蒙哥馬利乘法器之抵抗簡單及差動能量攻擊法的 RSA 密碼核心

學生：林祐進　　　　　　　　　　　　　　　　　指導教授：張錫嘉

國立交通大學電子工程學系　電子研究所碩士班

摘　　　要

這篇論文中介紹了一個使用可擴展蒙哥馬利模數乘法器的 RSA 密碼系統，這個可擴展的乘法器可以用在 GF(p)和 GF(2^m)並減少了 48%的運算時間相較於先前的架構。使用這個可擴展的乘法器的 RSA 密碼核心，最多可以支援 4096 位元任意長度。使用 TSMC .18 μm 設計流程實現這個架構後，所使用的邏輯閘數目為 365k，在 100MHz 的操作時脈下，完成金鑰長度為 4096 位元的 RSA 運算總共花費 355ms。此外考慮能量攻擊的防禦，分別使用隱藏私密金鑰及提出平衡能量消耗的方法，使得 RSA 密碼核心能夠抵抗簡單及差動能量攻擊。

A RSA Crypto-Core using Scalable Montgomery Multiplication with DPA and SPA Resistance

student：Yu-Ching Lin Advisors：Hsie-Chia Chang

Department of Electronics Engineering & Institute of Electronics
National Chiao Tung University

ABSTRACT

A scalable hardware architecture of modular multiplier on RSA cryptosystem is introduced in this thesis. The proposed scalable radix-2 multiplier is suitable for either GF(p) or GF(2n) and reduces 48% of computation time in contrast to previous scalable architectures. Based on the scalable multiplier, the proposed RSA crypto-core can work in any precision less than 4096 bits. After implemented by TSMC .18 μm technology, the gate count is 365k and a RSA encryption with 4096-bit key length can be completed in 355 ms under 100MHz operation. Furthermore, we also consider the power attack issues and take countermeasure to against DPA(differential power analysis) and SPA(simple power analysis) by blinding the secrect key and balanceing the power consumption.

誌　　　　謝

能夠完成這篇論文，指導教授：張錫嘉博士是功不可沒的，在作研究的路上，當我倦怠時能夠推我一把，當我徬徨時能夠指引我方向，才能夠順利完成研究所的學業及研究。也感謝林建青學長，對於 security 團隊(已正名為 STAR)的用心指導，因為學長的細心規劃及親身領導，讓整個團隊日益茁壯。柏均及人偉，謝謝你們在研究上給我這麼多建議，透過討論我才知道有甚麼不足及需要改進的地方，未來 STAR 就靠你們了。再來是並肩作戰的好夥伴大嘴，多虧有你的幫忙，才能讓我這麼有效率的完成論文；多虧有你開車溫暖接送，一群人一起去吃宵夜，才能撐過與研究論文奮鬥的夜晚。修齊學長也給了很多的幫助，尤其在口試投影片上的真知灼見，讓我能夠從挫咧蛋變成順利完成口試。張鑫偉感謝你在碩三的這幾個月先幫我墊房租，讓我不致於挨餓受凍，能夠專心做研究。李永裕，雖然你講話很靠盃，但是在一成不變的研究生活中，點綴一些歡樂。再次感謝所有實驗室的同學們。

最後要感謝總是在背後默默支持的家人及朋友們，爸媽有你們的支持才能讓我無後顧之憂的完成研究所的學業；有大學同學及朋友們的鼓勵，我才能堅持到最後一刻。再次感謝這一路上陪我成長茁壯的所有人。

A RSA Crypto-Core using Scalable Montgomery Multiplication with DPA and SPA Resistance

Student: Yu-Ching Lin

Advisor: Dr. Hsie-Chia Chang

Department of Electronics Engineering
National Chiao Tung University

Abstract

A scalable hardware architecture of modular multiplier on RSA cryptosystem is introduced in this thesis. The proposed scalable radix-2 multiplier is suitable for either $GF(p)$ or $GF(2^n)$ and reduces 48% of computation time in contrast to previous scalable architectures. Based on the scalable multiplier, the proposed RSA crypto-core can work in any precision less than 4096 bits. After implemented by TSMC .18 μm technology, the gate count is 365k and a RSA encryption with 4096-bit key length can be completed in 355 ms under 100MHz operation. Furthermore, we also consider the power attack issues and take countermeasure to against DPA(differential power analysis) and SPA(simple power analysis) by blinding the secrect key and balanceing the power consumption.

Contents

List of Figures

List of Tables

Chapter 1

introduction

1.1 Background

Since public key cryptosystem [1] was published in 1976 by Whitfield Diffie and Martin Hellman, the use of discrete logarithm problem in public-key cryptosystems has been recognized. This method of exponential-key exchange, which came to be known as Diffie-Hellman key exchange, was the first published practical method for establishing a shared secret-key over an unprotected communications channel without using a prior shared secret.

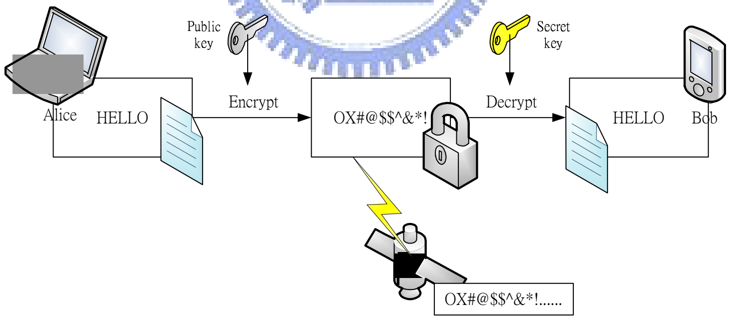


Figure 1.1: Public key system.

RSA and El-Gamal are two of the popular public-key cyrptosystems widely used nowadays. The RSA algorithm based on the high difficulty of factoring large numbers was published by Rivest, Shamir and Adleman [2] at MIT[1] in 1978. Further, the El-Gamal algorithm based on Diffie-Hellman key agreement describes the public-key system and

[1]Massachusetts Institute of Technology, located in Cambridge, MA, USA. http://web.mit.edu/

digital signature schemes, and it was proposed by Taher ElGamal [3] in 1985.

Figure 1.1 shows a scheme of public key system. In an encryption scheme anyone can encrypt using the public key, but only the holder of the private key can decrypt. Security depends on the secrecy of the private key.

The RSA is the most popular and well-defined security primary technique. It is a cryptosystem widely used to ensure data privacy in many fields such as communication. And also PKCS#1 standard [4] lines out a way of encrypting data using the RSA cryptosystem. Moreover, in digital signature and digital envelope, RSA provides non-repudiation and confidentiality of communication. Actually, many good security protocols using RSA cryptosystem are applied in the modern information technology, for example, virtual private networks, electronic commerce, and secure Internet access.

RSA cryptosystem is easy to understand and implement. It is based on modular exponentiation. This modular exponentiation is performed by repeated modular multiplications. In general, the modular multiplication has to be performed a certain number of times depends on the key length to ensure security, but the consequence is that the RSA operation has to take much more computational cost for security consideration. In order to include RSA cryptosystem practically in many protocols for high speed application, it is desired to devise faster encryption and decryption operations.

1.2 Motivation

In recent years, security issues on communications are more and more significant as the wireless industry explodes. The public key cryptosystem has become an important role. There are many applications using RSA as authentication for transactions and encryption or signature for secure messaging. The precision of operands is getting higher for better security. A major design concern for multiplication units used in cryptography is the large number of operand bits, 4096 bits in RSA, which causes large fanout of signals, large wire delays, and complex routing.

In this thesis, an approach provided to compute the modular multiplication in $GF(p)$, and word-based method can solve th high fanout problem. Furthermore, the precision of operand is limited only by the memory. In this thesis, any length less than 4168 bits can be performed and the architecture is so-called *scalable*.

Recent researches showed that power consumption may reveal the secret key of public key cryptosystem. Those attacks works based on the statistic analysis of power tracing. It's essentially to do something against the power attack. We also proposed a method on the RSA cryptocore that resists DPA and SPA.

1.3 Thesis Organization

In this thesis, a scalable RSA cryptosystem is given. The algorithms in this thesis are over prime field. The Montgomery multiplication algorithm over $GF(2^n)$ with radix-2 is given in the Chapter A since the algorithms of binary field with radix-2 are almost identical in digital system. . In Chapter 2, the preliminary mathematical background of RSA is first introduced and then we describe the RSA algorithm. In the end of this section, the Montgomery multiplication is introduced. In Chapter 3,an algorithm for word-based Montgomery modular multiplication over prime field is proposed. Also a brief introduction of power attack on modular exponentiation is given in the end. In Chapter 4.2, all the proposed scalable RSA cryptocore and word-based multiplier architectures are described in this chapter. First, the implementation of the proposed word-based Montgomery multiplication algorithm is presented. Then a comparison of several configurations of word-based Montgomery multiplication is described. The modular multiplication is the main operation for RSA scheme. The rest of section states the countermeasures of DPA and SPA. In Chapter 5, it shows the hardware implementation results and comparisons for ASIC and FPGA. Finally, the conclusion is given in Chapter 6.

Chapter 2

RSA Cryptosystem and Montgomery Multiplication

2.1 Mathematics Foundation

This chapter describes the basic arithmetic used in RSA cryttosystem over $GF(p)$. The most important mathematical tool is number theory, especially the theory of congruences. Modular arithmetic such as modular multiplication is especially an important part in the RSA systems, so there are still many approaches to its improvement nowadays.

2.1.1 Number Theory

Congruences

One of the most basic and useful in number theory is modular arithmetic, or congruences. Let a, b, n be integers with $n \neq 0$. If a and b differ by a multiple of n, a is congruent to b mod n.

$$a \equiv b(mod\ n)$$

It can be rewritten as

$$a \equiv b + nk$$

for some integer k.

Primitive Roots

In general, when p is a prime, a primitive root mod p is a number whose powers yield every nonzero class mod p. There are $\phi(p-1)$ primitive roots mod p. Let g be a primitive root for the prime p.

- If i is an integer, then $g^i \equiv 1(mod\ p)$ if and only if $i \equiv 0(modp - 1)$.
- If j and k are integers, then $g^j \equiv g^k(mod\ p)$ if and only if $j \equiv k(mod\ p - 1)$.

Fermat's Theorem

Fermat's theorem states the follows : If p is prime and a is a positive integer not divisible by p, then

$$a^{p-1} \equiv 1\ mod\ p \tag{2.1}$$

We know that if all of the elements of Z_p, where Z_p is the set of integers $\{0,1,\ldots,p-1\}$, are multiplied by a, modulo p, the result consists of all of the elements of Z_p in some sequence. Furthermore, $a \times 0 \equiv 0\ mod\ p$. Therefore, the $(p - 1)$ numbers

$$\{a\ mod\ p,\ 2a\ mod\ p, \ldots, (p-1)a\ mod\ p\}$$

are just the numbers $\{0,\ 1, \ldots,\ p - 1\}$ in some order. Multiplying the numbers in both sets and taking the result modulo p yields

$$1 \times 2 \times \ldots \times (p-1) \equiv (a\ mod\ p) \times (2a\ mod\ p) \times \ldots \times ((p-1)a\ mod\ p)$$
$$(p-1)!\ mod\ p \equiv (p-1)!a^{p-1}.$$

We can cancel the $(p-1)!$ term because it is relatively prime to p. This yields Equation 2.1.

Euler's Totient Function

Before presenting Euler's theorem, we need to introduce an important quantity in number theory, referred to as Euler's totient function and written $\phi(n)$, where $\phi(n)$ is the number of positive integers less than n and relatively prime to n. It should be clear that for a prime number p, $\phi(p) = p - 1$ There are two prime numbers p and q, with $p \neq q$. Then, for $n = pq$,

$$\phi(n) = \phi(pq) = \phi(p)\phi(q) = (p-1)(q-1). \tag{2.2}$$

Euler's Theorem

Euler's theorem states that for every a and n that are relatively prime:

$$a^{\phi(n)} \equiv 1 \ mod \ n \tag{2.3}$$

Equation 2.3 is true if n is prime, because in that case $\phi(n) = (n-1)$ and Fermat's theorem holds. However, it also holds for any integer n. Recall that $\phi(n)$ is the number of positive integers less than n that are relatively prime to n. Consider the set of such integers, labeled as follows:

$$R = x_1, x_2, \ldots, x_{\phi(n)}.$$

Now multiply each element by a modulo n:

$$S = (ax_1 \ mod \ n), (ax_2 \ mod \ n), \ldots, (ax_{\phi(n)}).$$

The set S is a permutation of R, by the following line of reasoning:

1. Because a and x_i are relatively prime to n, ax_i must also be relatively prime to n. Thus, all the elements of S are integers less than n that are relatively prime to n.

2. There are no duplicates in S. If $ax_i \ mod \ n = ax_j \ mod \ n$, then $x_i = x_j$.

Therefore,

$$\prod_{i=1}^{\phi(n)} ax_i (mod \ n) = \prod_{i=1}^{\phi(n)} x_i$$

$$\prod_{i=1}^{\phi(n)} ax_i \equiv \prod_{i=1}^{\phi(n)} x_i(\ mod \ n)$$

$$a^{\phi(n)} \prod_{i=1}^{\phi(n)} x_i \equiv \prod_{i=1}^{\phi(n)} x_i(\ mod \ n)$$

$$a^{\phi(n)} \equiv 1 \ mod \ n$$

An alternative form of the theorem is also useful:

$$a^{k\phi(n)+1} \equiv a \ mod \ n \tag{2.4}$$

2.1.2 N-residue

Given an integer $0 \leq a < n$, we define it's n-residue with respect to r as

$$\bar{a} \equiv a \cdot r \ mod \ n, \tag{2.5}$$

which is also called the Montgomery domain mapping.
It is easy to verify that

- $x + y$ is mapped to $\overline{x+y} = (x+y)r = xr + yr = \bar{x} + \bar{y}$,
- xy is mapped to $\overline{xy} = (xy)r = (xr)(yr)r^{-1} = \bar{x}\bar{y}r^{-1}$.

Therefore, the multiplication algorithm of the n-residue $\bar{x}$ and $\bar{y}$ is

$$\textbf{reduce}(\bar{x}\bar{y}).$$

The algorithms for subtraction, negation, equality test, inequality test,multiplication by an integer, and the greatest common divisor with n are also unchanged.

In order to map an integer x to it's n-residue $\bar{x}$ with respect to r, extra computation is required. Fortunately,the mapping operation can be done by

$$\bar{x} = \textbf{reduce}(xr^2) = xr^2r^{-1}mod \ n = xr \ mod \ n, \tag{2.6}$$

where $r^2 \ mod \ n$ need to be precomputed. And the inverse operation for mapping $\bar{x}$ to x is simply multiplied by 1.

$$\bar{x} = \textbf{reduce}(xr \cdot 1) = xr \cdot r^{-1}mod \ n = x \ mod \ n, \tag{2.7}$$

2.2 RSA Algorithm

The RSA scheme is the most widely used to ensure data privacy in many fields and applied to the digital signature generation and verification, the RSA DS algorithm, announced in ANSI[1] X9.31 [5]. It is a block cipher in which the plaintext and ciphertext are integers between 0 and $n - 1$ for some n which is typically between 2^{512} and 2^{4096}. The more bits provides the higher security. The scheme of RSA is showed as following:

[1]American National Standards Institute

Algorithm 2.1. ***(RSA Algorithm)***

Key generation

Select p,q	p *and* q *both prime,* $p \neq q$
Calculate N *and* $\phi(N)$	$N = pq$,$\phi(N) = (p-1)(q-1)$
Select integer E	$gcd(\phi(N), E) = 1$;$1 < E < \phi(N)$
Calculate D	$D \equiv E^{-1}$ *mod* $\phi(N)$
Public key	$KU = \{E, N\}$
Private key	$KR = \{D, N\}$

Encryption

Plaintext M	$M < N$
Ciphertext C	$C = M^E$ *mod* N

Decryption

Ciphertext C	$C < N$
Plaintext M	$M = C^D$ *mod* N
	$= M^{DE}$ *mod* N
	$= M$ *mod* N

2.2.1 RSA Rationale

Let p and q be two distinct large random primes. The modulus N is the product of these two primes: $N = pq$. According to equation(2.2), the Euler's totient function of N is given by

$$\phi(N) = (p-1)(q-1)$$

Now, select a number $1 < E < \phi(N)$ such that

$$gcd(\phi(N), E) = 1,$$

and compute D with

$$D \equiv E^{-1} mod\ \phi(N). \tag{2.8}$$

Here, $\{E, N\}$ is the public key and $\{D, N\}$ is the private key. The value of D and the prime numbers p and q are kept secret. Encryption is performed by computing

$$C = M^E\ mod\ N, \tag{2.9}$$

where M is the plaintext such that $0 \leq M < N$. The number C is the ciphertext from which the plaintext M can be computed using

$$M = C^D \; mod \; N. \tag{2.10}$$

The correctness of the RSA algorithm follow from Euler's theorem(Eq.2.3):
Let N and a be positive, relatively prime integers. Then

$$a^{\phi(N)} \equiv 1 \; mod \; N$$

Since ED is equal to $1 \; mod \; \phi(N)$, it meet that ED is equal to $1+k\phi(N)$ for some integer k.

$$\begin{aligned} C^D &\equiv (M^E)^D \; mod \; N \\ &\equiv M^{ED} \; mod \; N \\ &\equiv M^{1+k\phi(n)} \; mod \; N \\ &\equiv M^1 \times M^{\phi(N)^k} \; mod \; N \\ &\equiv M \times 1 \; mod \; N \end{aligned}$$

[Example]

Let $p = 47$ and $q = 59$, then $N = pq = 2773$ and $\phi(N) = (p-1)(q-1) = 2668$. The value of E must be chosen somewhere between 1 and 2668. Assume $E = 17$. According to Eq.(2.8), the value of D is 157. Assume further that the alphabet is represented by decimal values, i.e. $A = 01$, $B = 02$, $C = 03$, etc. and a blank space is given the value 00.
The plaintext ,M, is given as:

$$M = \text{RSA CRYPTOSYSTEM}$$

or represented in decimal as:

$$\begin{aligned} M &= (m_7 \; m_6 \; m_5 \; m_4 \; m_3 \; m_2 \; m_1 \; m_0) \\ &= (1819 \; 0100 \; 0318 \; 2516 \; 2015 \; 1925 \; 1920 \; 0513) \end{aligned}$$

The plaintext is encrypted block by block,which depends on the size of N, individually. As an example,the m_7 block is encrypted by

$$1819^{17}\ mod\ 2773 = 0818$$

Performing the same operation on the subsequent blocks generates an

$$c = 0818\ 1952\ 0578\ 2666\ 0774\ 0246\ 2109\ 0772.$$

Decrypting the message requires performing the same exponentiation using the decryption key $D = 157$, so

$$0818^{157}\ mod\ 2773 = 1819 = m_1$$

The rest of the plaintext can be recovered in this manner.

Note that the modular exponentiation is the most important operation in the RSA scheme.

2.2.2 Modular Exponentiation

In the RSA cryptosystem, the main operation for encryption and decryption is the modular exponentiation. The most direct way to compute $M^E\ mod\ N$ is to simply multiply M for E times. Since all the operands in RSA are typically large than 512 bits and it is impractical to store the result of M^E. There are alternative ways to make it efficient: the L-R binary method and the R-L binary method.
Supposed the key E is a n-bit in binary representation as:

$$E = (e_{n-1}, e_{n-2}, ..., e_2, e_1, e_0)_2,$$

then

$$M^E \bmod N = M^{(e_{n-1}\cdot 2^{n-1}+e_{n-2}\cdot 2^{n-2}+...+e_2\cdot 2^2+e_1\cdot 2^1+e_0)} \bmod N.$$

LR Method

$$M^E \bmod N \equiv ((\cdots(M^{e_{n-1}} \bmod N)^2 \cdots)^2 \cdot M^{e_1} \bmod N)^2 \cdot M^{e_0} \bmod N \qquad (2.11)$$

As showed in table, the L-R algorithm performs square and multiplication sequentially. It does mean that both the square and multiply operations can be performed in the same single hardware multiplier.

RL Method

$$M^E \bmod N \equiv (\cdots((M^{e_0} \bmod N)\cdot(M^2)^{e_1} \bmod N)\cdots)\cdot(M^{2^{n-1}})^{e_{n-1}} \bmod N \qquad (2.12)$$

In the R-L algorithm, the square and multiply operations are independent, and may be performed in parallel. Thus, 50% less clock cycles than LR algorithm are required to complete the exponentiation. However, two physical hardware multipliers are required to achieve this speed up.

Table 2.1: Compare of LR and RL algorithms

L-R algorithm	R-L algorithm
Input : n - bits $E = (e^{n-1}, \ldots, e^1, e^0)_2$	**Input :** n - bits $E = (e^{n-1}, \ldots, e^1, e^0)_2$
m - bits M	m - bits M
Output : m - bits Z	**Output :** m - bits Z
1. $P = M, Z = 1$	1. $P = M, Z = 1$
2. for $i = n-1$ to 0	2. for $i = 0$ to $n-1$
2.1 if $(e_i == 1)$	2.1 if $(e_i == 1)$
2.2 $Z = Z \cdot P \ mod \ N$;	2.2 $Z = Z \cdot P \ mod \ N$;
2.3 $Z = Z \cdot Z \ mod \ N$;	2.3 $P = P \cdot P \ mod \ N$;
3. return Z;	3. return Z;

2.3 Montgomery Multiplication

In 1985, P. L. Montgomery introduced an efficient algorithm for computing $R = ab \ mod \ n$ where a, b and n are k-bit binary numbers. The algorithm is particularly suitable for implementation on general-purpose computers and embedded microprocessors. The algorithm use divisions by a power of two, which are simply a few right shifts in hardware implementation, instead of trail divisions by n, which are used in a conventional modular operation. The basic idea of Montgomery's method, **reduce** algorithm, is stated as following.

2.3.1 Reduction Algorithm

The Montgomery reduction algorithm 2.2 computes the resulting k-bit number R without performing a division by the modulus n. Via an ingenious representation of the residue class modulo n, this algorithm replaces division by n operation with division by a power of 2. This operation can be easily accomplished on a computer since the numbers are represented in binary form. Assuming the modulus n is a k-bit number, i.e., $2^{k-1} < n < 2^k$, let r be 2^k. The Montgomery reduction algorithm requires that r and n be relatively prime, i.e., $gcd(\ r\ ,\ n) = gcd(\ 2^k\ ,\ n\ ) = 1$. This requirement is satisfied if n is odd. The Montgomery reduction algorithm 2.2 was showed as following.

Since $gcd(r, n) = 1$, there are two numbers r^{-1} and n' with $0 < r^{-1} < n$ and $0 < n' < r$, satisfying

$$rr^{-1} - nn' = 1 \tag{2.13}$$

Algorithm 2.2. ***(reduce(x))***

```
Input : $x$,$r$,$n$
Output: $a = xr^{-1} mod\ n$
  1. $q = (x\ mod\ r)n'\ mod\ r$;
  2. $a = (x + qn)/r$;
  3. If $a > n$, then $a = a - n$;
  4. return $a$;
```

The reason why the Algorithm 2.2 works is explained as follows.
First,

$$xr^{-1} = xrr^{-1}/r = x(nn' + 1)/r. \tag{2.14}$$

For any integer l,

$$\begin{aligned}((xn' + lr)n + x)/r \ (mod\ n) &= (xn'n + lrn + x)/r \ (mod\ n) \\ &= (xn'n + x)/r \ (mod\ n)\end{aligned}$$

Therefore, instead of computing $q = xn'$, we can compute $q = xn' \ mod\ r$. Supposed that $0 \leq x < rn$,the value of $a = (x + qn)/r$ will be less than $2n$. Therefore,computing $a\ mod\ n$ can be done by simply subtracting n from a if $a > n$.

2.3.2 Montgomery Multiplication Algorithm

When the numbers X,Y,and N are large, we can apply the above Algorithm (2.2) to compute $XY\ mod\ N$ in an efficient way. We want to compute $Z = XY\beta_{-1}$ and suppose that

$$X = \sum_{k=0}^{m-1} x_k\beta^k, Y = \sum_{k=0}^{m-1} y_k\beta^k, Z = \sum_{k=0}^{m-1} z_k\beta^k, N = \sum_{k=0}^{m-1} n_k\beta^k.$$

The Montgomery multiplication is showed as follows:

Algorithm 2.3. ***(Montgomery** (X,Y,N))*

Input : X,Y,N

Output: $Z = XY\beta^{-m} mod\ N$

```
1. $Z = 0$;
2. for $k = 0$ to $m - 1$
    2.1 $q = (Z + x_kY)(\beta - n_0)_\beta^{-1} mod\ \beta$ ;
    2.2 $Z = Z + x_kY + qN$;
    2.3 $Z = Z/\beta$;
3. return $Z$;
```

In the Algorithm 2.3, we use the notation $(\alpha)_\beta^{-1}$ to denote the inverse of α in Z_β. The reason why the algorithm works is explained as follows. The algorithm computes the answer Z incrementally. At each iteration, x_k is scanned to determined the operation. The computation is similar to the **reduce** Algorithm 2.2. That is, compute $Z = Z + x_kY + qN$ and then $Z = Z/\beta$. The rationale behind this computation is to find a proper value of q

so that, at the k-th iteration, the value of $Z + x_kY + qN$ is a multiple of β. As explained above, the value of q is

$$q = (Z + x_kY)N^{'} mod\ \beta,$$

where $N^{'}$ is the inverse of N in Z_{β^m}. Therefore,

$$q = (Z + x_kY)N^{'} mod\ \beta = (z_0 + x_ky_0)(\beta - n_0)_{\beta}^{-1} mod\ \beta,$$

In the above equation, we use the fact that

$$RR^{-1} - NN^{'} \equiv 1 \pmod{\beta^m}$$

since $gcd(N, \beta^m) = 1$, where $R = \beta^m$.
Thus, $-NN^{'} mod\ \beta = 1$, which implies that

$$N^{'} mod\ \beta = (-N)_{\beta}^{-1} = (\beta - n_0)_{\beta}^{-1}.$$

Suppose that $\beta = 2^k$ for some positive integer k, and that the value of $n_0 = \beta - 1$. Then the value of $(\beta - n_0)_{\beta}^{-1} = 1$. In this case, $q = (z_0 + x_ky_0)\ mod\ \beta$, which is the value of the last digit of $Z + x_kY$. Therefore, we can save the computation of the value of q at each iteration. Supposed that $k = 1$, the Montgomery Multiplication algorithm with radix-2 can be rewritten as following algorithm 2.4.

Algorithm 2.4. ***(Montgomery with radix-2)***
Input : $X = (x_{m-1}, \ldots, x_0)_2, Y = (y_{m-1}, \ldots, y_0)_2, N = (n_{m-1}, \ldots, n_0)_2$
Output: $Z = (z_{m-1}, \ldots, z_0)_2$

```
1. $Z = 0$;
2. for $k = 0$ to $m - 1$
    2.1 $q = Z + x_kY$;
    2.2 $q_0 = q\ mod\ 2$;
    2.3 $Z = q + q_0N$;
    2.4 $Z = Z/2$;
3. return $Z$;
```

2.4 Power Analysis of Modular Exponentiation

Cryptographers have traditionally analysed cipher systems by modeling cryptographic algorithms as ideal mathematical objects. Conventional techniques such as differential [6] and linear [7] cryptanalysis are very useful for exploring weaknesses in algorithms. But the physical implementations often result in the leakage of side-channel information.

Attacks have been proposed that use such information as timing measurements [8], power consumption [9], electromagnetic emissions and faulty hardware. In this section we examine the weakness of RSA cryptographic algorithms to power analysis attacks. Specifically, attacks on the modular exponentiation process are described.

Power analysis attacks work by exploiting the differences in power consumption between when a tamper-resistant device processes a logical zero and when it processes a logical one. For example, when the secret data on a smartcard is accessed, the power consumption may be different depending on the Hamming weight of the data. If an attacker knows the Hamming weight of the secret key the attacker could potentially learn the entire secret key. This type of attack, where the adversary directly uses a power consumption signal to obtain information about the secret key is referred to as a Simple Power Analysis (SPA) attack and is described in section 2.4.1. Differential Power Analysis (DPA) is described in section 2.4.2 and it is based on the same underlying principle of an SPA attack, but uses statistical analysis techniques to extract very tiny differences in power consumption signals.

2.4.1 Simple Power Attack (SPA)

An SPA attack, as described in [9], involves directly observing a system's power consumption. Suppose that the attackers not only have unlimited access, but also have detailed knowledge of the software and hardware of the systems. If an attacker can determine where certain instructions are being executed, it can be relatively simple to extract useful information.

SPA on a single-key cryptographic algorithm, such as DES, could be used to learn the Hamming weight of the key bytes. DES uses only a 56-bit key so learning the Hamming weight information alone makes DES vulnerable to a brute-force attack. In fact, depending on the implementation, there are even stronger SPA attacks. A two-key, public-key cryptosystem, such as an RSA or elliptic curve cryptosystem, might also be vulnerable to an SPA attack on the Hamming weight of the individual key bytes, however it is possible an even stronger attack can be made directly against the square-and-multiply algorithm.

If exponentiation were performed in software using one of the square-and-multiply algorithms, there could be a number of potential vulnerabilities. The main problem with both algorithms is that the outcome of the "‘if statement"’ might be observed in the power signal. This would directly enable the attacker to learn every bit of the secret exponent. A simple fix is to always perform a multiply and to only save the result if the exponent bit is an one. This solution is very costly for performance and still may be vulnerable if the act of saving the result can be observed in the power signal.

2.4.2 Differential Power Attack (DPA)

A DPA attack is more powerful than an SPA attack because the attacker does not need to know as many details about how the algorithm was implemented. The technique also gains strength by using statistical analysis to help recover side-channel information.

The problem with an SPA attack is that the information about the secret key is difficult to directly observe. The information about the key was often obscured with noise and modulated by the device's clock signal. DPA can be used to reduce the noise and also to "‘demodulate"’ the data. Any power biases at the time corresponding to the guess bit operation are visible as an obvious spike in the difference signal and much of the noise is eliminated because averaging reduces the noise variance.

Three attacks of DPA, was described in [10]. Other assumptions used for particular attacks are stated in the following that describe the specific attack details.

Single-Exponent, Multiple-Data (SEMD) Attack

The SEMD attack assumes that the smartcard is willing to exponentiate an arbitrary number of random values with two exponents: the secret exponent and a public exponent. The basic attack is that by comparing the power signal of an exponentiation using a known exponent to a power signal using an unknown exponent, the adversary can learn where the two exponents differ, thus learn the secret exponent. In reality, the comparison is nontrivial because the intermediate data results of the square-and-multiply algorithm cause widely varying changes in the power signals, thereby making direct comparisons unreliable. The solution to this problem is to use averaging and subtraction.

Multiple-Exponent, Single-Data (MESD) Attack

The MESD attack is more powerful than the SEMD attack. The SEMD attack is a very simple attack requiring little sophistication on the part of the adversary, but the resulting DPA bias signal is sometimes difficult to interpret. The Signal-to-Noise Ratio (SNR) can be improved using the MESD attack. The assumption for the MESD attack is that the smartcard will exponentiate a constant value using exponents chosen by the attacker. This value may or may not be known to the attacker.

Zero-Exponent, Multiple-Data (ZEMD) Attack

The ZEMD attack is similar to the MESD attack, but has a different set of assumptions. One assumption for the ZEMD attack is that the smartcard will exponentiate many random messages using the secret exponent. This attack does not require the adversary know any exponents, hence the zero-exponent nomenclature. Instead, the adversary needs to be able to predict the intermediate results of the square-and-multiply algorithm using an off-line simulation. This usually requires that the adversary know the algorithm being used by the exponentiation hardware and the modulus used for the exponentiation. There are only a few common approaches to implementing modular exponentiation algorithms, so it is likely an adversary can determine this information. It is also likely that the adversary can learn the modulus because this information is usually public.

2.4.3 Countermeasure of RSA Against DPA and SPA

Potential countermeasures to the attacks described in this paper include many of the same techniques described to prevent timing attacks on exponentiation. Kocher's [8] suggestion for adapting the techniques used for blinding signatures can also be applied to prevent power analysis attacks. Prior to exponentiation, the message could be blinded with a random value, v_i and unblinded after exponentiation with $v_f = (v_i^{-1})^e mod\ N$. An efficient way is presented in [8] to calculate and maintain (v_i, v_f) pairs.

Message blinding would prevent the MESD and ZESD attacks, but since the same exponent is being used, the SEMD attack would still be effective. To prevent the SEMD attack, exponent blinding would be necessary. In an RSA cryptosystem, the exponent can be blinded by adding a random multiple of $\phi(N) = (p-1)(q-1)$, where and $N = pq$. Note that $M^{\phi(N)}\ mod\ N \equiv 1\ mod\ N$, the result of exponentiation is unchanged since

$$\begin{aligned} M^{k\cdot\phi(n)+E}\ mod\ N &\equiv M^{k\cdot\phi(N)} \times M^E\ mod\ N \\ &\equiv (M^{\phi(N)})^k \times M^E\ mod\ N \\ &\equiv (1)^k \times M^E\ mod\ N \\ &\equiv M^E\ mod\ N \qquad (2.15) \end{aligned}$$

,where k is a random number.

Chapter 3

Proposed Montgomery Multiplication

3.1 Review of Montgomery Multiplication Algorithms

Since the Montgomery multiplication(MM) algorithm was proposed in 1985, various modular multiplication [11], [12], [13], [14], [15] were modified based on it. The MM is the basic operation used in modular exponentiation, which is required in the Diffie-Hellman and RSA public-key cryptosystems. Recently, the implementations of Montgomery Multiplication are focused on the elliptic curve cryptography [16] over the finite fields $GF(p)$ and $GF(2^m)$. Following section shows some of the modified algorithm and gives a brief comment of each algorithms.

Algorithm 3.1. ***(Chen's Modified Montgomery Multiplication)***

Input : X,Y,N

Output: Z

```
1. $C = X \times Y = c_{2n-1}2^{2n-1} + c_{2n-2}2^{2n-2} + \ldots + c_1 2^1 + c_0 2^0$
2. $Z = 0$;
3. for $k = 0$ to $2n - 1$
     3.1 $q = (z_0 + c_k) mod\ 2$ ;
     3.2 $Z = Z + c_k Y + qN$;
     3.3 $Z = Z/2$;
```

```
4. return $Z$;
```

In Chen's algorithm, the multiplication, $X \times Y$ is computed before the loop and the loop is the same as reduce algorithm 2.2. For hardware implementation with pipeline, modular operation can work without waiting the final result of $X \times Y$. This means that multiplication and modulus can be work in parallel. The disadvantage of Chen's algorithm is the number of iteration that is *two times* than Montgomery's. However, there is *only one* n-bit addition in each iteration in Chen's, since the computation of q is simply bitwise XORed.

Algorithm 3.2. ***(Yang's Modified Montgomery Multiplication)***
Input : X,Y,N
Output: Z

```
1. $C = X \times Y = c_{2n-1}2^{2n-1} + c_{2n-2}2^{2n-2} + \ldots + c_1 2^1 + c_0 2^0$
      $= C_U \times 2^n + C_L$
2. $Z = 0$;
3. for $k = 0$ to $n-1$
     3.1 $q = (z_0 + c_k) mod\ 2$ ;
     3.2 $Z = Z + c_k + qN$;
     3.3 $Z = Z/2$;
4. $Z = Z + C_U$
5. return $Z$;
```

Yang [11] proposed an algorithm in 1998 that was improved to the Chen's algorithm by separating C into upper part C_U and lower part C_L, thus the iterations are half of Chen's regardless of the final addition. The rationale is explained below.

$$\begin{aligned} A \times B \times 2^{-n}\ mod\ N &\equiv C \times 2^{-n}\ mod\ N \\ &\equiv (C_U \times 2^n + C_L) \times 2^{-n}\ mod\ N \\ &\equiv (C_U + C_L \times 2^{-n})\ mod\ N \end{aligned}$$

The [15] was implemented based on Yang's algorithm with pipelined enhancement.

3.2 Word-based Montgomery Multiplication Algorithm

Nowadays, a major design concern for multiplication units used in cryptography is the large number of operand bits, which causes large fanout of signals, large wire delays, and complex routing. These problems are reduced in systolic architectures [13], [14], at the cost of extra hardware resources. However, these architectures are usually tailored for fixed-precision.

Tenca and Koc proposed a scalable word-based architecture [17] based on radix-2 Montgomery Multiplication. Differently from the high-radix algorithms used in software, [17] avoids the use of costly digit multiplications, this way, it allows the exploration of several design trade offs to obtain the best performance in a limited chip area, without limiting the operand precision. Practical limits to the precision are imposed by the control and memory subsystems.

Algorithm 3.3. ***(Word-based Montgomery Multiplication Algorithm)***

Input : X,Y,N

Output: S

```
1. $Z = 0$;
2. for $i = 0$ to $n-1$
    2.1 $(C_a, Z^0) = x_i Y^0 + Z^0$ ;
    2.2 if $Z_0^0 = 1$ then;
        i. $(C_b, Z^0) = Z^0 + N^0$ ;
        ii. for $j = 1$ to $e$;
            A. $(C_a, Z^j) = C_a + x_i Y^j + Z^j$ ;
            B. $(C_b, Z^j) = C_b + N^j + Z^j$;
            C. $Z^{j-1} = (Z_0^j, Z_{w-1:1}^{j-1})$;
    2.3 else;
        i. for $j = 1$ to $e$;
            A. $(C_a, Z^j) = C_a + x_i Y^j + Z^j$ ;
            B. $Z^{j-1} = (Z_0^j, Z_{w-1:1}^{j-1})$;
3. return $Z$;
```

Algorithm 3.3 executes a series of operation to generate $XYr^{-1} mod\ N$, scanning Y and N word-by-word and scanning X bit-by-bit. This characteristic enables us to derive a hardware implementation that is very regular and based on simple operations. Suppose that n is a multiple of word size w, the n-bit operands are split into e words, where $e = n/w$. Word and bit vectors are represented as:

$$N = (0, N^{e-1}, \ldots, N^1, N^0),$$
$$Y = (0, Y^{e-1}, \ldots, Y^1, Y^0),$$
$$Z = (0, Z^{e-1}, \ldots, Z^1, Z^0),$$
$$X = (x_{n-1}, \ldots, x_1, x_0),$$

where the word are marked with superscripts and the bits are marked with subscripts. N, Y, and Z are zero-extend to $e + 1$ words, thus avoiding overflow. The concatenation of two vectors A and B is represented as (A,B). The bit position i of the kth word of an operand A is represented as A_i^k.

The total carry-out value generated in each j loop iteration corresponds to $C_a + C_b$ and it is in the range [0,2]. The algorithm computed a new partial sum of Z for each bit, x_i, scanning the words of the present Z, Y, and N. Once Y is completely scanned, next bit of X, x_{i+1}, is taken and then scan is repeated. The arithmetic operations in the j loop are performed in w bits of precision. The number of loop iterations is adjusted to accomplish the required precision, without modifications to the inner structure. This is the main feature of word-based scalable architecture shown in [17].

The right-shift by 1 bit is done by $Z^{j-1} = (Z_0^j, Z_{w-1:1}^{j-1})$. Note that the least significant bit of next word, Z_0^j, must be computed before it can be right-shifted into the most significant position of Z^{j-1} on the jth step of the inner loop. This is a critical limitation of the algorithm.

To implement this algorithm, unrolling the for loop to pipelined architecture is a useful skill for increment of parallelism. The dependency on the carry bits within j loop restricts their parallel execution. However, instructions in different i loops may be executed in parallel.

3.3 Modified Word-based Montgomery Multiplication

The fundamental problem with the Tenca-koc architecture is the dependency caused by waiting to right shift Z_0^j into Z_{w-1}^{j-1} before processing Z^{j-1} at the next iteration. The work in [18] describes that rather than shift right the partial result Z^k, shift left the Y and N can eliminate the dependency between Z^k.

Algorithm 3.4. ***(Proposed Modified Word-based Montgomery Multiplication)***
Input : X,Y,N
Output: Z

```
1. $Z = 0$;
2. for $i = 0$ to $ker-1$
    2.1 for $j = 0$ to $p-1$
        i. $Y = 2 \times Y$;
        ii. $N = 2 \times N$;
        iii. for $k = 0$ to $e + \lceil p/w \rceil - 1$
            A. $(C_a, Z^k) = C_a + x_{(ip+j)}(Y^k) + Z^k$ ;
            B. if$(k == \lfloor \frac{j}{w} \rfloor)$, $odd = S^{\lfloor \frac{j}{w} \rfloor}_{(j \ mod \ w)}$;
            C. $(C_b, Z^k) = C_b + odd(N^k) + Z^k$;
        iv. $Z = S/Z^p$;
3. return $Z$;
```

In this algorithm, i loop and j loop scans the x bit-by-bit and k loop computes a new partial sum Z word by word for preset bit of X. Set n=ker$\cdot p$,thus scanning through n bits of X. The number of iteration in k loop is $e + \lceil p/w \rceil$,where $e = n/w$ is the number of words. Note that there are additional $\lceil p/w \rceil$ iterations to compute the most significant word caused by the left-shift of Y and N.

Since $Y < N < 2^n$, it is easy to verify that

$$\begin{aligned}(x_0 2^0 Y + odd_0 2^0 N) + (x_1 2^1 Y + odd_1 2^1 N) + \cdots + (x_{p-1} 2^{p-1} Y + odd_1 2^{p-1} N) \\ &< (2^0 Y + 2^0 N) + (2^1 Y + 2^1 N) + \cdots + (2^{p-1} Y + 2^{p-1} N) \\ &\leq (2^p - 1)(Y + N) \\ &< (2^p)(N + N) \\ &< (2^p)(2 \cdot 2^n) \\ &\leq 2^{p+n},\end{aligned}$$

,where the subscript of odd denotes the the number of loop been executed. Therefore, extra p bits or $\lceil p/w \rceil$ words are necessary to handle the overflow.

Note that the odd check is different from the [17], always check the least significant bit of partial sum Z_0^0. Unrolling technique is used again to the j loops, instead of shifting right partial sum in the k loop, we shift left Y and N for each j loop. The odd check must be modified to match the original one. It is observed that

$$\begin{aligned} jth\ Z^0 &= (j-1)th\ Z^0/2 + x_{j-1}Y \quad (PWBMM) \\ jth\ Z^0 &= (j-1)th\ Z^0 + x_{j-1}2Y \quad (MM), \end{aligned}$$

and the $jth\ S^0$ in this algorithm is a shift-left-1-bit version of [17]. And also the odd bit shift left by 1 bit for each loop. Therefore,

$$odd_0 = Z_0,\ odd_1 = Z_1, \ldots,\ odd_{j-1} = Z_{j-1}.$$

And the representation in word base is showed as:

$$Z_{(j\ mod\ w)}^{\lfloor \frac{j}{w} \rfloor},$$

where j is the number of loop and w denotes the word size.

The rest of section states the evaluation of equality of this algorithm to original algorithm.

Original Montgomery Multiplication can be represented as:

$$XY2^{-n} \ mod \ N$$

$$\equiv \frac{\frac{\frac{\frac{x_0Y+odd_0N}{2}+x_1Y+odd_1N}{2}+\cdots+x_{n-2}Y+odd_{n-2}N}{2}+x_{n-1}Y+odd_{n-1}N}{2} \tag{3.1}$$

$$\equiv \frac{x_0Y+odd_0N}{2^n}+\frac{x_1Y+odd_1N}{2^{n-1}}+\cdots+\frac{x_{n-2}Y+odd_{n-2}N}{2^2}+\frac{x_{n-1}Y+odd_{n-1}N}{2}. \tag{3.2}$$

My proposed algorithm PMWBMM is showed as:

$$\overbrace{\frac{\frac{\frac{(x_0 2^0 Y+odd_0 2^0 N)+(x_1 2^1 Y+odd_1 2^1 N)+\cdots+(x_p-12^{p-1}Y+odd_{p-1}2^{p-1}N)}{2^p}+(x_p 2^0 Y+odd_p 2^0 N)+\cdots}{2^p}+\cdots}{2^p}}^{ker=n/p} \tag{3.3}$$

$$= \frac{(\frac{x_0Y+odd_0N}{2^p}+\frac{x_1Y+odd_1N}{2^{p-1}}+\cdots+\frac{x_p-1Y+odd_{p-1}N}{2})+x_pY+odd_pN}{2^p}+\frac{x_{p+1}Y+odd_{p+1}N}{2^{P-1}}+\cdots \tag{3.4}$$

$$= \frac{x_0Y+odd_0N}{(2^p)\cdot(2^p)^{ker-1}}+\frac{x_1Y+odd_1N}{2^{p-1}(2^p)^{ker-1}}+\cdots+\frac{x_{n-1}+odd_{n-1}N}{2}. \tag{3.5}$$

Since ker=n/p, it is sufficient that equation(3.5) equals to equation(3.2).

Chapter 4

Proposed RSA Crypto-Core

4.1 Overall Architecture

Back to the equation 2.6, equation 2.7, and the modular exponentiation algorithm mentioned in subsection 2.1.2 and subsection 2.2.2. The mapping operation is necessary before and after the Montgorery Multiplication. In the thesis, we choose R-L method in order to increase the parallelism, thus 2 sets of multiplier are required.

Figure 4.2 shows a block diagram of the proposed RSA kernel core. There are four 4096-bit registers,Z, P, N, and the key E. And two 32*64-bit proposed word-based Montgomery Multipliers are used to execute multiplication and square in parallel.

Figure 4.1 shows a flow chart of the algorithm. The $r^2 \ mod \ N$ is precomputed as an input, and stored in the Z-reg in the FETCH STATE. In the same state, we also store plaintext, M, in P-reg, modulus, N, in N-reg and 64 bits of key, E. In the PRE MM STATE, the operands must map to their N-residue, multiplying by $r^2 \ mod \ N$. Note that, in the R-L algorithm, Z is initialized to 1, and the mapping is MM(1,Z) since $Z = r^2$. The proposed multiplier supports multiplying by 1. It is helpful for N-residue mapping and save the hardware cost by sharing the Z-reg to store r^2. The next flow is DET STATE, scanning the key bit-by-bit and multiplying Z by P if the scanned bit is 1. The POST MM STATE is reached when the key is completely scanned. The ciphertext is valid after the result of N-residue is transformed back to integer.

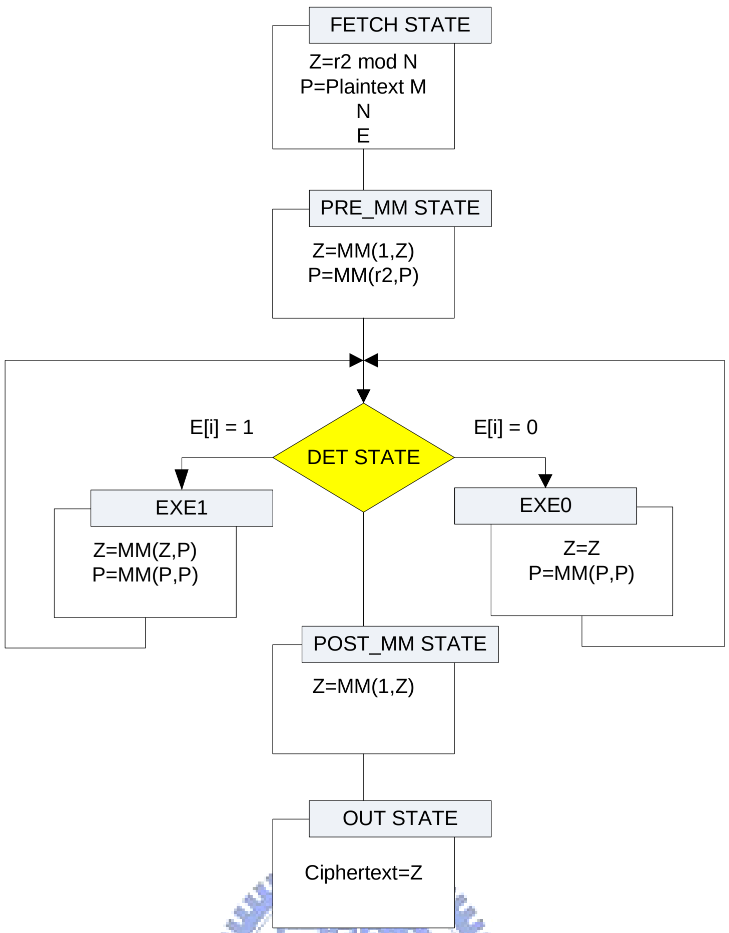


Figure 4.1: RSA flow chart.

4.2 Modular Multiplier Architectures

The proposed architecture of the reconfigurable multiplier with DPA resistant is presented in this chapter. As mentioned in subsection 3.2, the precision of operands is only limited by the memory size and control subsystems. In this thesis, it is adapted to three precision, 1024, 2048, and 4096 bits, over the prime fields $GF(p)$. All of the required materials for mathematical theorems have been mentioned in early chapters. Then all of these main components used in the scalable multiplier are detailed in the following subsections.

4.2.1 A Scalable Montgomery Multiplier

Back to the algorithm 3.4 which is mentioned in subsection 3.3. Due to he final subtraction may cause the leakage of power attack, it is necessary to remove the conditional

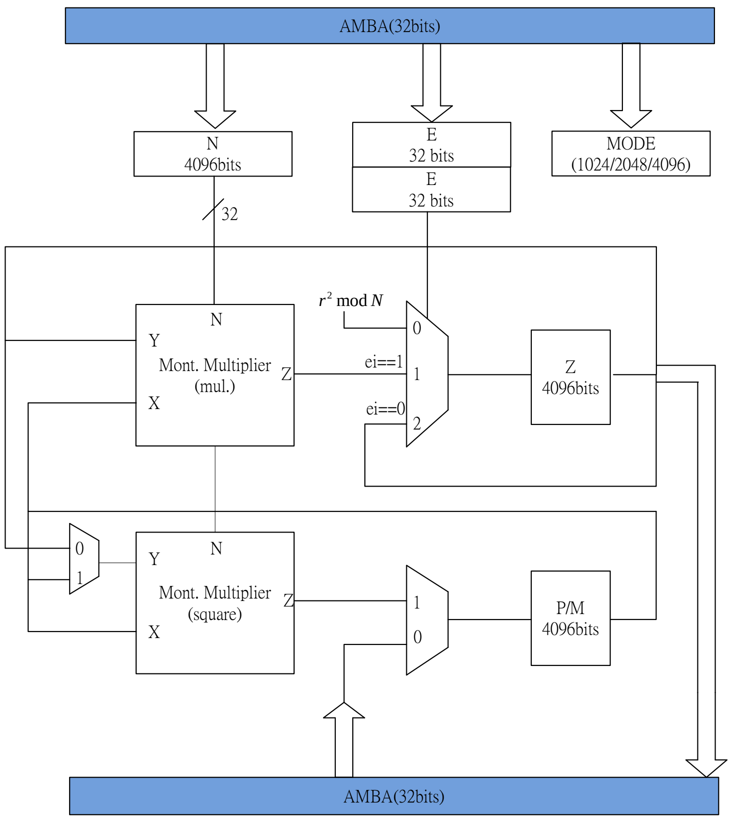


Figure 4.2: RSA modular exponentiation architecture.

statements from the algorithm. Eliminating the final subtraction can be done by exercising a few iterations without extra cost in hardware [19]. The shortcoming is that increasing one more word and one more kernel cycle of latency.

The multiplications by 2 of Y and N are integrated into the word-based operation. The proposed algorithm is modified as following and easy for hardware mapping.

Algorithm 4.1. ***(Proposed Modified Word-based Montgomery Multiplication)***

Input : X,Y,N

Output: S

```
1. $S = 0$;
2. for $i = 0$ to $ker-1$
    2.1 for $j = 0$ to $p-1$
        i. for $k = 1$ to $e$
            A. $(C_a, S^k) = C_a + x_{(ip+j)}(Y^k_{w-2:0}, Y^{k-1}_{w-1}) + S^k$ ;
            B. $odd = S^{\lfloor \frac{j}{w} \rfloor}_{(j \ mod \ w)}$;
        C. $(C_b, S^k) = C_b + odd(N^k_{w-2:0}, N^{k-1}_{w-1}) + S^k$;
    ii. $S = S_{n+p-1:p}$;
3. return S;
```

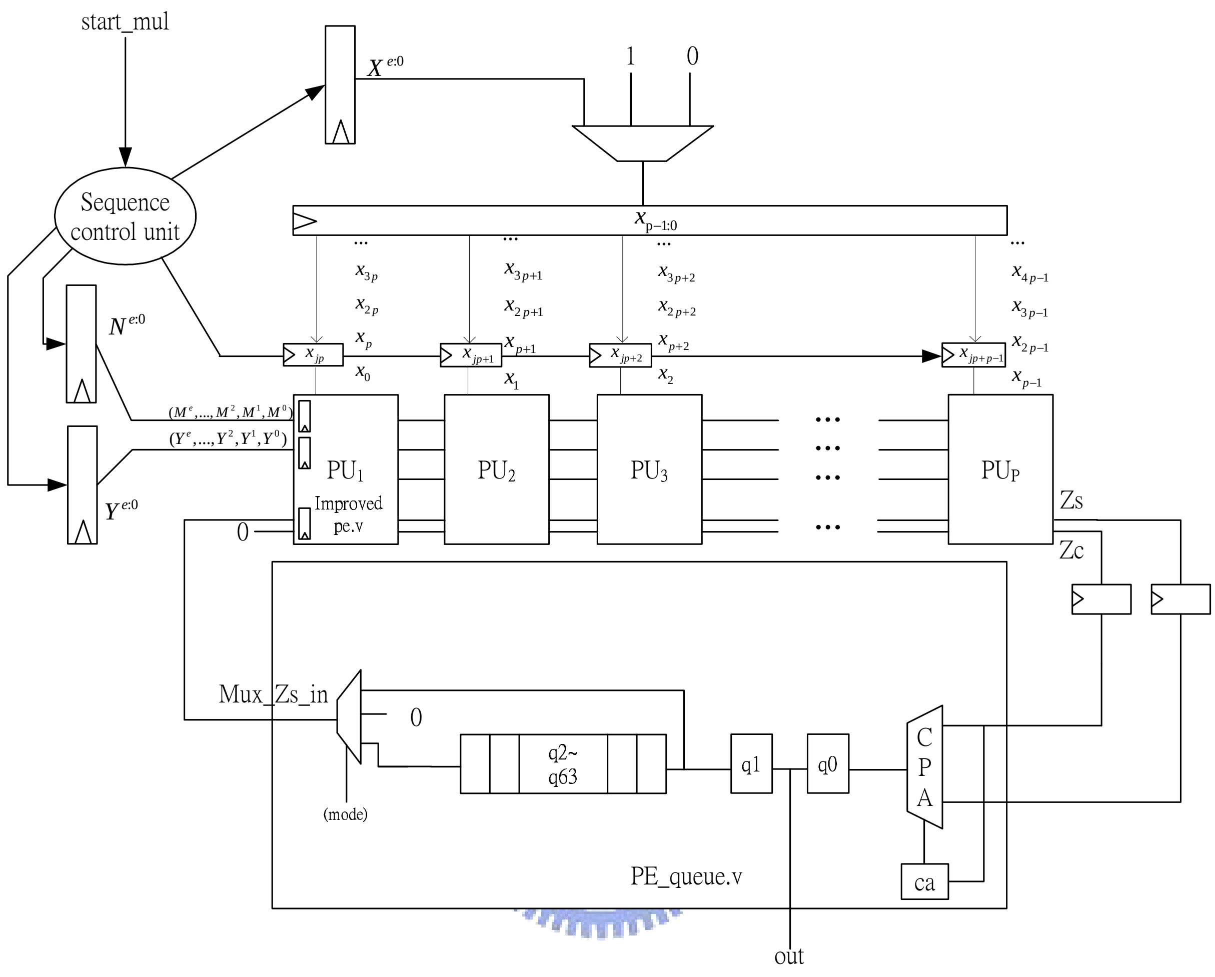


Figure 4.3: Scalable word-based Montgomery Multiplier.

Figure 4.3 shows a block diagram of the scalable word-based Montgomery Multiplier. The j loop in proposed algorithm is unrolled and mapped to a pipelined kernel of p w-bit processing units (PUs). The result of partial sum S is in carry save representation(CSR), requiring $2w$ bits, thus reducing the critical path between PUs. PU1 can process the next bit of X unless the result of PUP is ready or PU1 itself has finished the words.

For higher precision application, PUP may compute the S before PU1 has finished the additional words. The results must be queued until PU1 is available again. The clock cycles for one PU to handle one bit of X is called *kernel cycle*. For large operands, the queue in CSR consumes significant area, so we convert S to nonredundant form using

w-bit CPA. Also the nonredundant S is an output of the multiplication. In the i loop ,ker= n/p kernel cycle are required to the entire computation, scanning n bits of X. The detail relationship between w, p, n, is described in the next section.

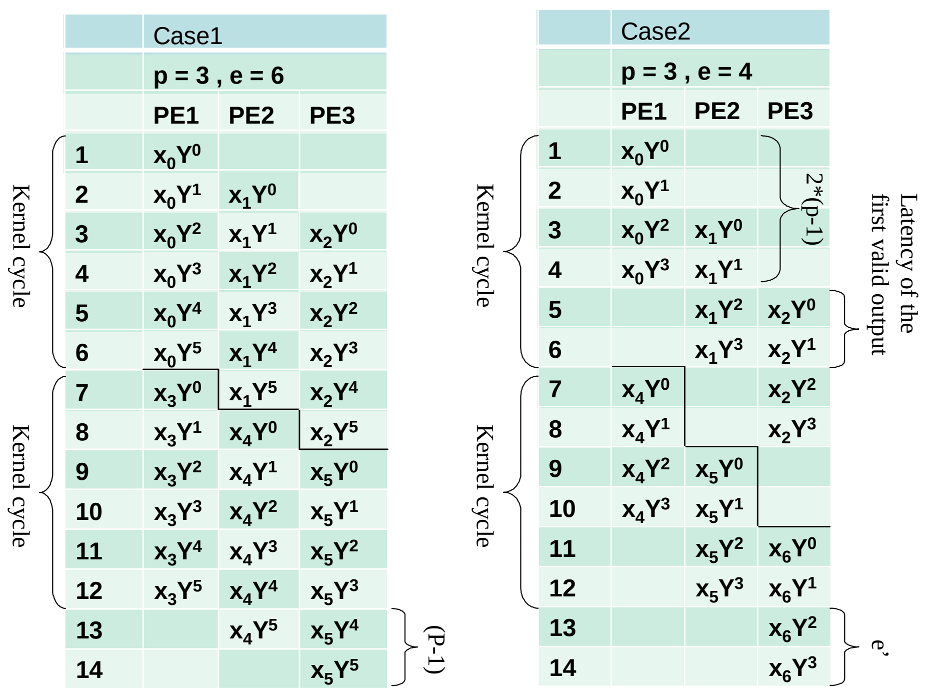
4.2.2 Number of Processing Unit and Size of Word

The time to compute n bits depends on the word-size w and number of PUs p. The kernel cycle is stated in Table 4.1. Figure 4.4 shows a pipelined diagram for providing better understanding of the computation time. The e in this figure is the total number of words but not n/p. The PU1 can't start computation of next bit of X until the PU1 completes e words computation or the last PUP has computed the first valid output word.

Table 4.1: Kernel cycle and computation time.

	Tenca-Koc	This work
The first valid output of PUP(eq1)	$2(p-1)+2= 2p$	$(p-1)+(\lceil p/w \rceil + 1)$ $= p + \lceil p/w \rceil$
Number of words to be processed(eq2)	$e+1$	$e + \lceil p/w \rceil$
computation time (eq1>eq2)	$\text{ker}(2p)+((e+1)-2)$	$\text{ker}(p + \lceil p/w \rceil)+(e + \lceil p/w \rceil - (\lceil p/w \rceil + 1))$
computation time (eq1≤eq2)	$\text{ker}(e+1)+2(p-1)$	$\text{ker}(e+\lceil p/w \rceil)+(p-1)$

Tenca-Koc [17]
There are 2-cycle latency because of the dependency between S^j and $Sj-1$. The case1 corresponds to large number of words, and the word number dominates the kernel cycle. There are $e+1$ clock cycles in the kernel cycle. There ker kernel cycles. Finally, $2(p-1)$ cycles are required for the subsequent PUs to complete the last kernel cycle. The case2 corresponds to a small number of words. Each kernel cycle takes $2p$ clock cycles before final PU takes 2 cycles to produces its first valid word. Finally, $(e+1)-2$ cycles are required to obtain the rest of words at the end of the last kernel cycle.



	Case1		
	p = 3 , e = 6		
	PE1	PE2	PE3
1	x_0Y^0		
2	x_0Y^1	x_1Y^0	
3	x_0Y^2	x_1Y^1	x_2Y^0
4	x_0Y^3	x_1Y^2	x_2Y^1
5	x_0Y^4	x_1Y^3	x_2Y^2
6	x_0Y^5	x_1Y^4	x_2Y^3
7	x_3Y^0	x_1Y^5	x_2Y^4
8	x_3Y^1	x_4Y^0	x_2Y^5
9	x_3Y^2	x_4Y^1	x_5Y^0
10	x_3Y^3	x_4Y^2	x_5Y^1
11	x_3Y^4	x_4Y^3	x_5Y^2
12	x_3Y^5	x_4Y^4	x_5Y^3
13		x_4Y^5	x_5Y^4
14			x_5Y^5

	Case2		
	p = 3 , e = 4		
	PE1	PE2	PE3
1	x_0Y^0		
2	x_0Y^1		
3	x_0Y^2	x_1Y^0	
4	x_0Y^3	x_1Y^1	
5		x_1Y^2	x_2Y^0
6		x_1Y^3	x_2Y^1
7	x_4Y^0		x_2Y^2
8	x_4Y^1		x_2Y^3
9	x_4Y^2	x_5Y^0	
10	x_4Y^3	x_5Y^1	
11		x_5Y^2	x_6Y^0
12		x_5Y^3	x_6Y^1
13			x_6Y^2
14			x_6Y^3

Figure 4.4: Pipelined diagram.

Proposed Architecture

The proposed architecture reduce the latency to only 1 clock cycle. For case1, there are $e + \lceil p/w \rceil$ clock cycles in the kernel cycle. $(p - 1)$ cycles are required to complete the final kernel cycle. The improvement is slightly better or even a little bit worse because the system is still limited by the time for PU1 to complete. The case2 takes p clock cycles before final PU takes $\lceil p/w \rceil + 1$ cycles to produces its first valid word in each kernel cycle. Finally, $e + \lceil p/w \rceil - (\lceil p/w \rceil + 1)$ cycle are required to obtain the more significant words at the end of the last kernel cycle. In this case, the system is limited by the time for PUP to complete and speed up significantly.

The queue size is also related to the w, n and p. In case2, PU1 starts the next kernel cycle after PUP computes the valid output immediately so there is no need for any queue. Case1, the output of PUP must be queued until PU1 completes current kernel cycle. Total size of queue is (total word - first valid output)$\times w$-bit.

Figure 4.5 shows the computation time for a Montgomery multiplication of precision n of several wp configurations. The more detail table is showed in Appendix. Observe that, when the operand precision is small, the number of PUs may be small and, when the precision is high, the number of PUs should be as high as possible. Thus, the final decision on the actual configuration depends on the precision for which the hardware will be used the most and the available area. In my proposed architecture, there are 64 PUs and the word size is 32 bits. Table 4.2 shows a comparison of computation time on this configuration. The optimal configuration of 4096-bit implementation is $wp = 4096$. However, the proposed architecture make the trade-off on area limited by FPGA and throughput.

Table 4.2: A comparison of computation time

	w=32 and p=64	
	Tenca-Koc [17]	This work
n=1024	2079	1087
n=2048	4159	2175
n=4096	8382	8351

4.2.3 Processing Unit

Figure 4.6 shows the architecture of the processing unit. There are 2 w-bit carry save adders to do the redundant arithmetic.

$$S_s, (ca, S_c) = S_s + x_i \cdot 2Y + (S_c, ca)$$
$$S_s, (cb, S_c) = S_s + odd \cdot 2N + (S_c, cb)$$

The registers, 1-bit ca and cb, can store the carry and be added to the next word as a carry-in. Since the carry out is stored by the carry-reg, the S_c only uses $w-1$ bits. And the S_s is w bits. The registers, N^{-1} and Y^{-1}, are here to store the present MSB and to implement the left-shift by 1. In one PU, the MSB of previous N^{k-1}_{w-1} in the N^{-1}-reg is concatenated with the present $N^{k}_{w-2:0}$

$$(N^{k}_{w-2:0}, N^{k-1}_{w-1}),$$

and becomes the input of the next PU.

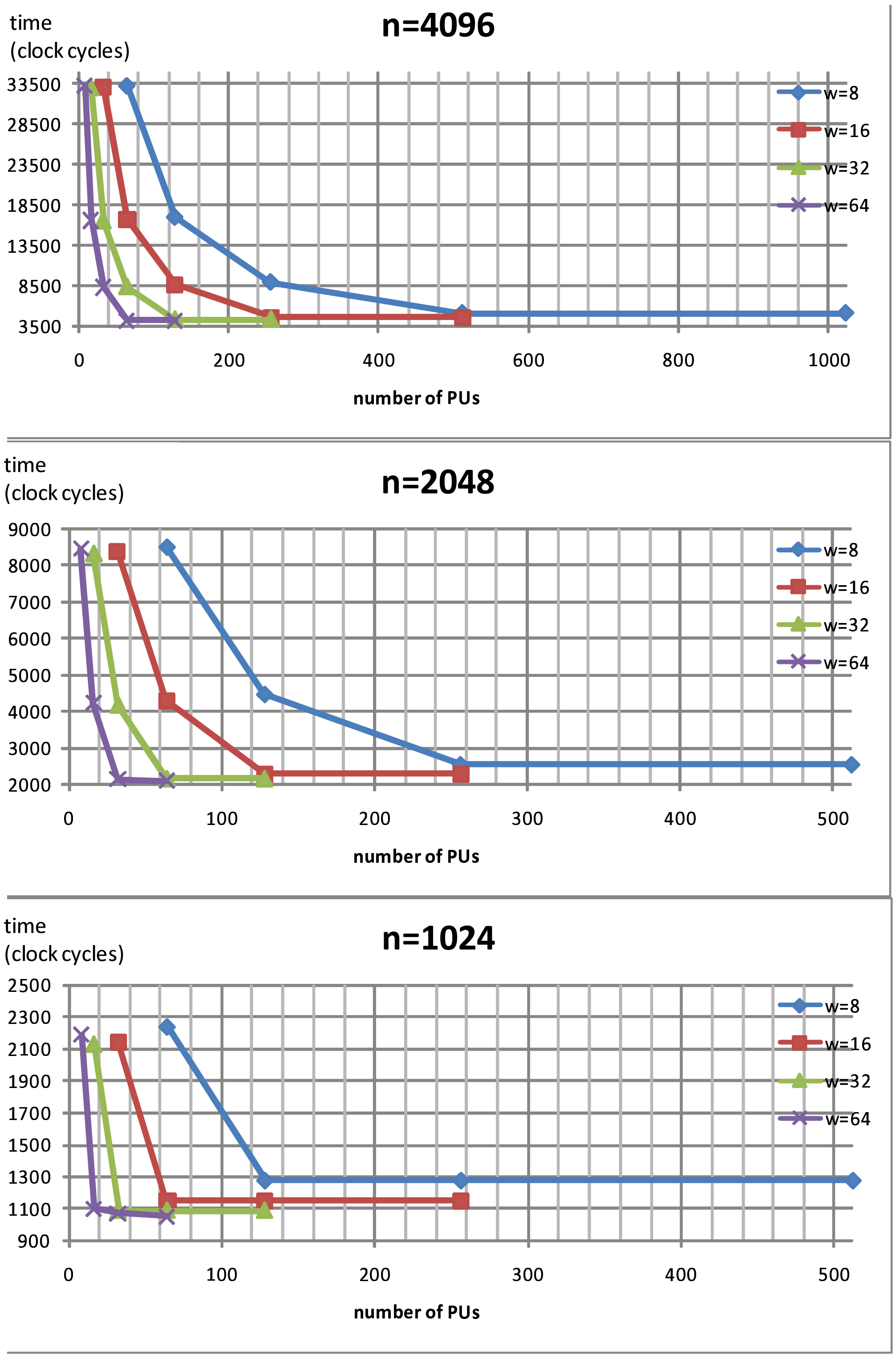


Figure 4.5: Several configurations of n=1024, 2048, and 4096.

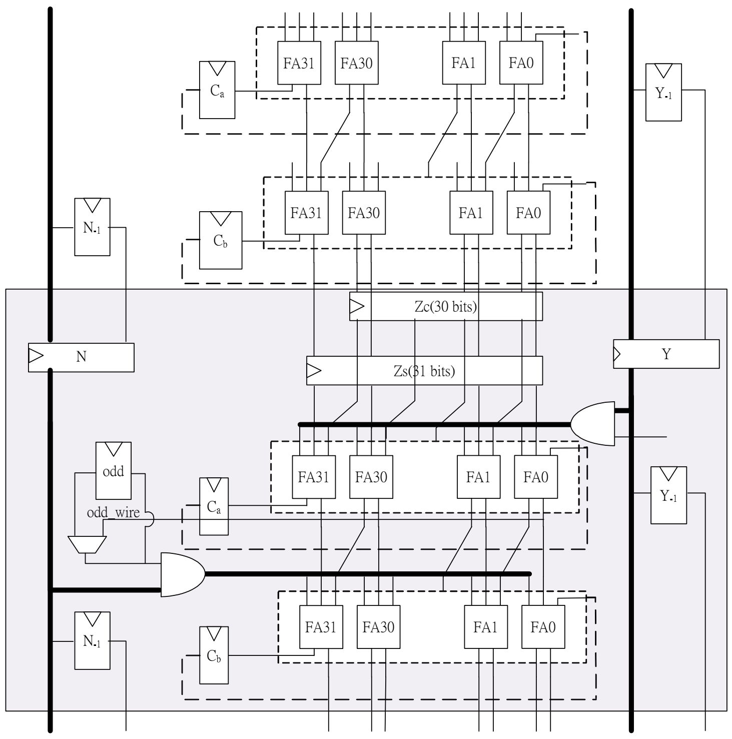


Figure 4.6: Architecture of Processing Unit

The odd parity is determined in the first word, and then stored in the register named odd. However, the PU cannot use the value stored in the odd reg to processing the first word. Therefore, the odd check is done by odd-reg or directly the odd-wire depends on whether PU is processing the first word. Note that the odd-check bits are distinct from PUs.

$$odd = S_{j \ mod \ w}^{\lfloor \frac{j}{w} \rfloor}$$

4.2.4 Flexible Output and Permutation Function

Preventing final subtraction in Montgomery multiplication is accomplished by enlarging the bit-size of operands. If the length is a multiple of 64, the output is the sum of last PUs. Otherwise, it is costly for performance to have output fixed at the last PUs especially when the length is short. Figure 4.7 shows the architecture of permutation function which can output at arbitrary PU depending on the performed length. The details of permutation function is showed in Appendix C. Therefore, the multiplication can be finished as soon as the overall length is scanned instead of waiting for the operands passed through 64 PUs.

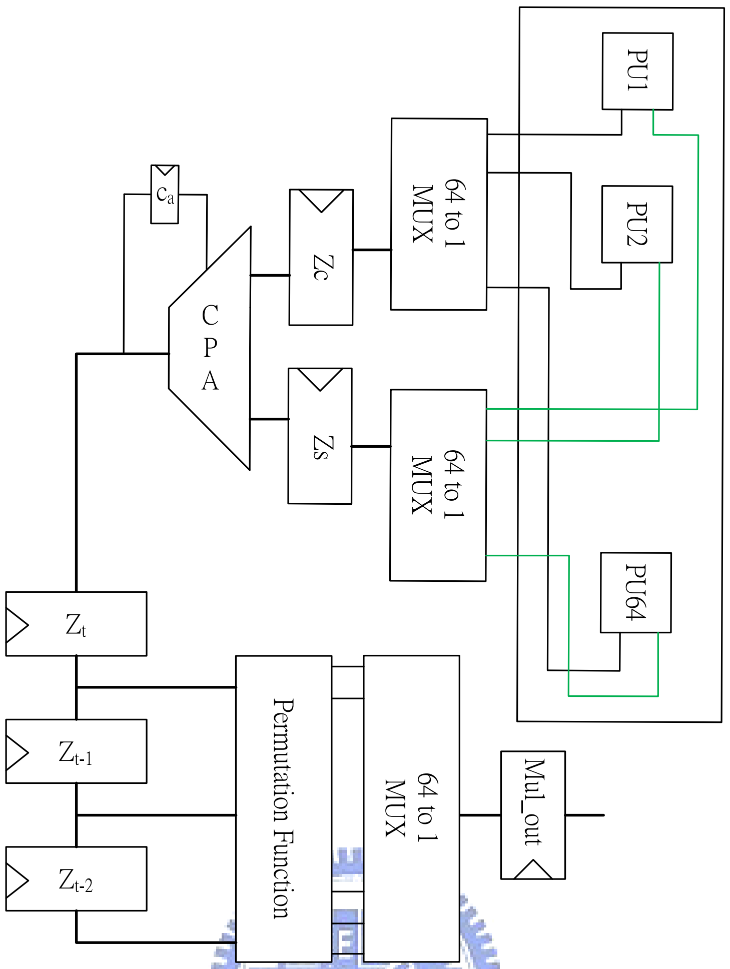


Figure 4.7: Architecture of Flexible Output.

4.3 Countermeasures Against DPA and SPA

The basic security of RSA is based on th difficulty of factoring the product of two primes. But recent research discovered that the information of the key can be estimated by tracing the power consumption. DPA is a powerful tool that allows cryptanalysis to extract secret keys and compromise the security of smart cards and other cryptographic devices by analyzing their power consumption. Simple Power Analysis (SPA) is a simpler form of the attack that does not require statistical analysis.

One work mentioned in [19] is a countermeasure against DPA because the final subtraction of output depends on the inputs. And also the output is related to the key. In the thesis, the proposed architecture of 2 multipliers consumes the same power since that the 2 multipliers always compute despite of the bit of key. The only difference is that Z-reg keeps its value when E^i is 0, thus cause the weakness of SPA.

Since the power dissipation depends on the operation of registers, it is reasonable to make the operations identical. The proposed countermeasure redesign a register that can be read/written in the manners of shifting or index-addressing. The way of operation is controlled by an input random signal, producing by a hardware PRNG[1] or in software. Every time one multiplication is completed, the next configuration of the operation of registers is set randomly. Therefore, the power consumption is always the same. The proposed RSA cryptocore can resist the SPA attack.

Recall the equation 2.15 in section 2.4.2, $M^{e+r\cdot\phi(n)} \equiv M^e \ mod\ n$. The DPA countermeasure can be done by randomly generating a integer r and adding $r\cdot\phi(N)$to the original key E as a new key. Therefore, the key guessed by the adversary is randomized thus preventing the ZEMD attack.

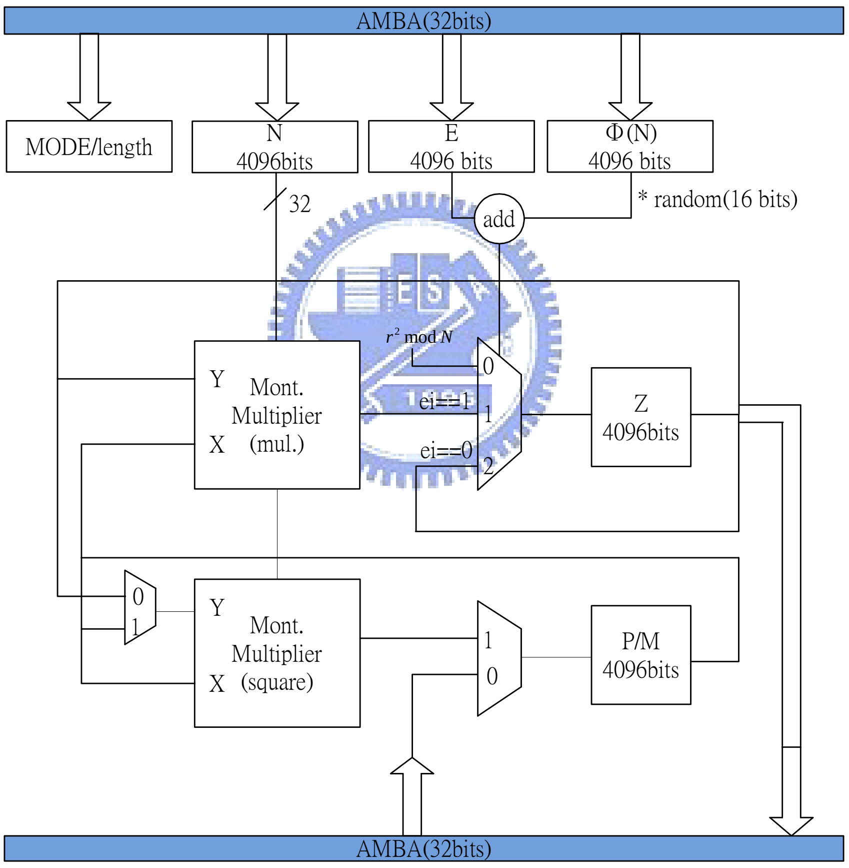


Figure 4.8: RSA modular exponentiation architecture against DPA.

In the proposed design, we set r is 16-bit and the modular multiplication is always executed $n+16+2$ times during encryption. This is the overhead of DPA countermeasure.

[1]Pseudo random number generator.

There are some assumptions when ZEMD power attack is mounted on the RSA scheme, the first z bits,$\{e^{z-1}, \cdots, e^0\}$, are known by the attacker. The attacker can guess the next bit, e^z, by analysis of the power trace at the $z+1$ iteration. In this work, the key always varies with the integer r. The power information is useless for the attacker, thus the design is DPA resistant.

Figure 4.8 shows the modified version of Fig 4.2,which preventing DPA attack. In contract to the design without DPA countermeasure, an additional 4096-bit is to store $\phi(N)$ and a 16-bit PRNG is to generate the random number r. The original key E is also fully loaded at the beginning, and the random key is computed word-by-word. Every time a word of key has been performed, the next word of key E^i is added to the $r \cdot \phi(N)$. The random number r is kept unchanged until a RSA exponentiation is completed. Finally, there are total 5 4096-bit Flip-flops, X, Y, N, E and $\phi(N)$ respectively, in the proposed architecture.

Chapter 5

Implementation Results and Comparison

A word-based RSA scheme in both software and hardware are given in this work. This chapter shows the hardware implementation results. The software simulation environment is constructed in C programing languages and the optimization level is O3. The execution time of software manner over various precisions is listed below.

Table 5.1: Modular exponentiation software performance.

Field	$GF(P_{1024})$	$GF(P_{2048})$	$GF(P_{4096})$
Time (s)	0.49	16.26	24.86
Throughput (kb/s)	2.1	0.167	0.164

In this thesis, all of the design in hardware is implemented using RTL (Register-Transfer-Level) Verilog HDL (hardware description language) and synthesized on both application-specific integrated circuit (ASIC) and field-programmable gate arrays (FP-GAs). The technology of ASIC design is using TSMC[1] 0.18μm CMOS process and the technology of FPGA design is using Xilinx[2] Virtex-4 xc4vlx160 platform FPGAs. The RTL synthesizer uses Synopsys[3] Design Compiler for ASIC and Xilinx ISE for FPGA.

[1]Taiwan Semiconductor Manufacturing Company Ltd. http://www.tsmc.com/

[2]Xilinx, Inc. The developer and fabless manufacturer of FPGAs. http://www.xilinx.com

[3]Synopsys, Inc. http://www.synopsys.com/

5.1 ASIC Implementation

The logic synthesis is performed with RTL synthesizer uses Synopsys Design Compiler using TSMC 0.18μm CMOS standard-cell technologies. The data throughput of RSA is given by

$$\frac{n(\texttt{exercised precision})}{k(\texttt{efficient key length}) \times t_{MM}(\texttt{computation time of multiplication})}.$$

The clock frequency is set to 100MHz and gatecount is 365k. The detail value is shown as table 5.2

Table 5.2: The verification results on ASIC.

Design	ASIC		
Technology	TSMC 0.18μm		
Clock frequency	100MHz		
Gate count	365k		
Precision	1024	2048	4096
Computation time (ms)	12.7	47.8	355
Throughput (kb/s)	80	43	11.5

Since there are few implementations of 4096-bit RSA, table 5.3 shows the comparison with other 1024-bit RSA implementations with ASIC design. In contrast to proposed design, the work [20] shows a small area but the throughput is also slow. For the application of smart cards, it is suitable to reduce area. In contrast to [20], the throughput of proposed design is 2 times better for 2048 and 4096, but the area is only double. The computation time of this work is counted with the worst case, the efficient key length is the same as modulus, of each precision. The area of [21] is much higher than the others, since it is radix-2^{32}. But the throughput is also higher than any others.

5.2 FPGA Implementation

Table 5.4 shows the detail value of the proposed design. The clock frequency is set to 102 MHz and the total slices is 26879 include 3 n-bit F/Fs. Table 5.5 shows the comparison with other 1024-bit RSA implementations on FPGA. McIvor [22] uses 1024-stages PE to

Table 5.3: Comparison with other 1024-bits implementations with ASIC design.

Author	Chen [20]	Mukaida [21]	Proposed
Platform	.18μm CMOS	.18μm CMOS	.18μm CMOS
Combinational gatecount	37k	755k	146k
Register gatecount	138k	210k	218k
Frequency (Mhz)	370	200	200
Throughput (kb/s)	83	5000	162
Note	16 PEs*w=16 21 kb/s for 2048 5.4 kb/s for 4096	radix-2^{32}	64 PUs*w=32 include 3 n-bit registers

implement the RSA cryptosystem. The word length of each PE is one bit. McIvor's [22] another approach uses CRT(Chinese Remainder Theorem) to speed up the decryption. Tang [23] implement the RSA cryptosystem with radix-2^{17}.

Table 5.4: The verification results on FPGA.

Design	FPGA		
Technology	xc4vlx160		
Clock frequency	102 MHz		
Slices(include registers)	34331		
Slices(w/o registers)	26879		
Precision	1024	2048	4096
Throughput(kb/s)	81	40	9

Table 5.6 shows three implementations of the proposed architecture. The number of F/Fs and LUTs in Proposed2 contrast to Proposed1 is increased due to the extra 2 4096-bit registers. The F/Fs are almost the same between Proposed2 and Proposed3 since there are identical number of registers. But there are about increased 9000 LUTs caused by 3 64-to-1 multiplexers in flexible output.

Table 5.5: Comparison with other 1024-bits implementations on FPGA.

Author	Chen [20]	McIvor [22]	Tang [23]	Proposed
Platform	XC2V8000	XC2V6000	XC2V3000	xc4vlx160
Number of slices(w/o registers)	1673	N.A.	8190	26879
Number of slices(include registers)	6783	26136	14334	26879
Frequency (Mhz)	116.7	97.08	90	102
Throughput(kb/s)	26	376	429	81
Note	16PEs*w=16 multiplier		radix-2^{17}	64PUs*w=32 3 4096

Table 5.6: Comparison of 4096-bit implementations on FPGA.

Note	Proposed1	Proposed2 DPA	Proposed3 DPA/$GF(2^n)$/flexible out
Platform		xc4vlx160	
Number of F/Fs slices	34331	42651	43281
Number of LUTs	40458	57682	66617
Number of slices	26879	37688	40203
Frequency (Mhz)	102	104	106
Throughput(kb/s)	81	80.6	81.7
Note	3 4096-bit	5 4096-bit	5 4096-bit

Chapter 6

Conclusion

A total solution in hardware and software to the word-based scalable RSA cryptocore in $GF(p)$ is given in this thesis. In order to deal with various precision, $0 < n \leq 4168$, of operands, the word-based Montgomery techniques are employed. A Montgomery modular multiplication algorithm is proposed. The modular exponentiation is the main operation of RSA that exercising series of modular multiplications. The implementation of the proposed multiplier in this work shows a considerable trade-off on area and throughput. The cost of area is proportioned to $w \times p$, where w and p denote the size of word and number of PUs respectively. For high speed application, it can be modified to speedup 50% by doubling the PUs with a few alternations.

According to the implementation result, it is synthesized using .18μm CMOS technology with 365k gates and using Xilinx Virtex-4 xc4vlx160 with 26879 slices in FPGA design. It takes about 24.86 s to accomplish a 4096-bit RSA operation in software but takes only 355 ms in hardware. It is 70 times fast in throughput. The throughput of encryption is limited by the efficient key length of DPA, which is $n + 16$.

In the Appendix A, Algorithm A.1 shows that the radix-2 Montgomery multiplications over prime field $GF(p)$ and binary field $GF(2^n)$ are almost identical. The proposed design can be modified to support binary field $GF(2^n)$ operation by simply eliminating the carry. Therefore, the unified scalable multiplier can be used in ECC crytposystem.

Appendix A

Algorithm of Montgomery multiplication over $GF(2^n)$

The modular multiplication is also an important operation used in ECC cryptosystem. Since the ECC and RSA are widely used, we attempt to design a cryptographic processor containing above cryptosyatems. The proposed Montgomery multiplier can be shared by the en/decryption in RSA and ECC. Not only for prime field $GD(p)$, but also the modular multiplication is usually performed over binary field $GF(2^n)$ in ECC. The Montgomery multiplication over binary field is stated in this section.

Recall the algorithm 2.4 in section 2.3.2, it is the Montgomery multiplication over prime field $GF(p)$ with redix-2. The Montgomery multiplication algorithm for $GF(2^n)$ is given below:

Algorithm A.1. ***(montgomery multiplication over*** $GF(2^n)$ ***with radix-***2)

Input : *$a(x), b(x), p(x)$, and m*

Output: $c(x)$

```
1. $c(x) = 0$;
2. for $k = 0$ to $m - 1$
    2.1 $q(x) = (c_0(x) + a_k(x)b_0(x))p_0^{'}(x)$;
    2.2 $q_0 = q(x)$ mod $x$;
    2.3 $c(x) = (c(x) + a_k(x)c(x) + q(x)p(x)$;
    2.4 $c(x) = c(x)/x$;
3. return c(x);
```

where $p_0^{'}(x) = p_0^{-1}(x)\ mod\ x$. It can be seen that the two algorithms are almost identical except that addition operation in $GF(p)$ becomes a bit-wise modulo-2 addition in $GF(2^n)$. In proposed algorithm, the extra reduction step at the end is removed, since it is no necessary for $GF(2^n)$. Although the operands are integers in the form algorithm and binary polynomials in the latter, the representations of both are identical in digital systems. The division by x is also identical to division by 2 in digital systems. Therefore, both field modular multiplication can be implemented on the same hardware with a *Field_sel* signal to decide which field is performed.

Appendix B

FPGA Implementation with AMBA

In this thesis, since this work is mainly implemented on ASIC design, there is not any technique used to improve the performance on FPGA. Thus, the implementation results on FPGA is slightly worse in timing performance, but it is helpful in fast verification and gives reliable hardware information.

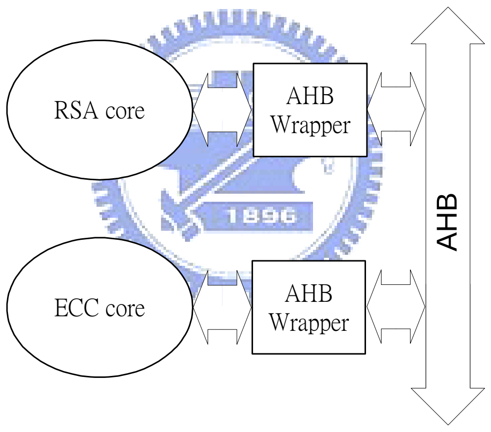


Figure B.1: RSA and other device connect to AHB via AHB wrapper.

The Advanced Microcontroller Bus Architecture [24] was introduced in 1996 and is widely used as the on-chip bus for ARM[1] processors. In its 2nd version, ARM introduced AHB that is a single clock-edge protocol. This protocol is today a standard for 32-bit embedded processors because it is well documented and can be used without royalties. AMBA is designed for use in System-on-a-chip (SoC) designs. The important aspect of a SoC is not only which components or blocks it houses, but also how they are intercon-

[1]ARM, Inc. http://www.arm.com/

nected. AMBA is a solution for the blocks to interface with each other. Figure **??** shows RSA core and ECC core are interconnected with each other on AHB.

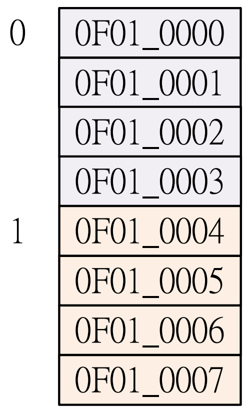


Figure B.2: Address Mapping.

Figure B.3 shows the integration of the proposed RSA core and the AHB wrapper that meets the AMBA protocol. The memory addresses are mapping to different devices on the bus. The Figure B.2 denotes that there is a 2-bit offset for 32-bit bus transfer since the memory is byte-addressed.

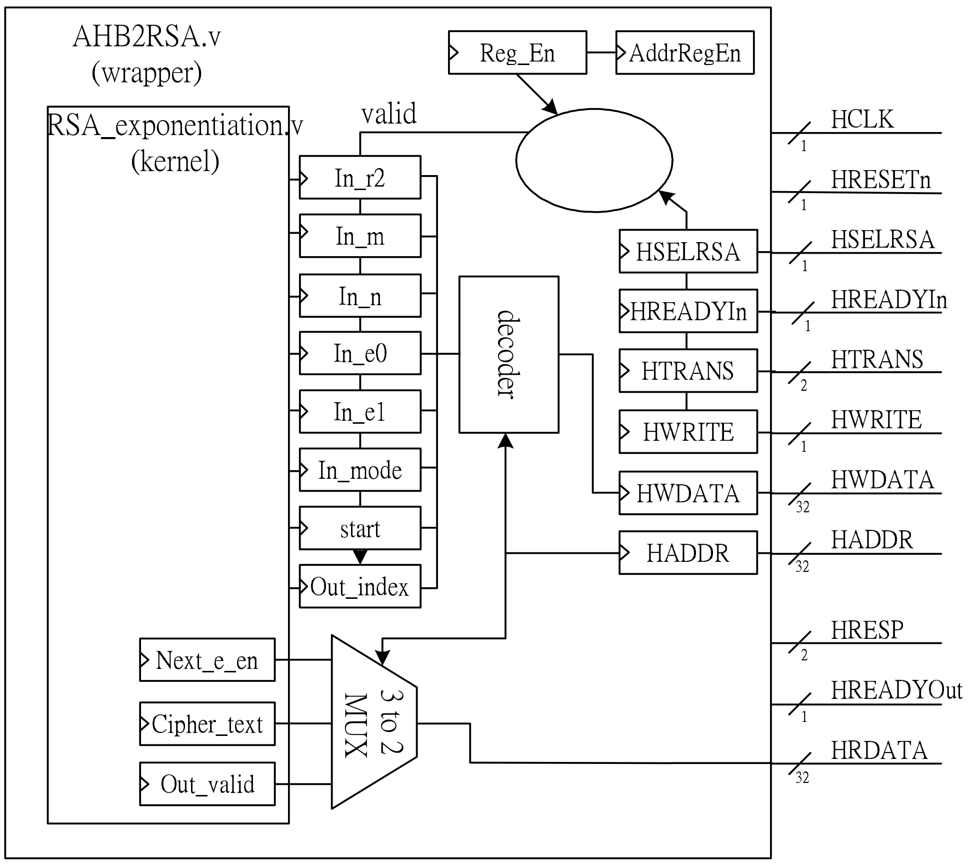


Figure B.3: AHB to RSA wrapper.

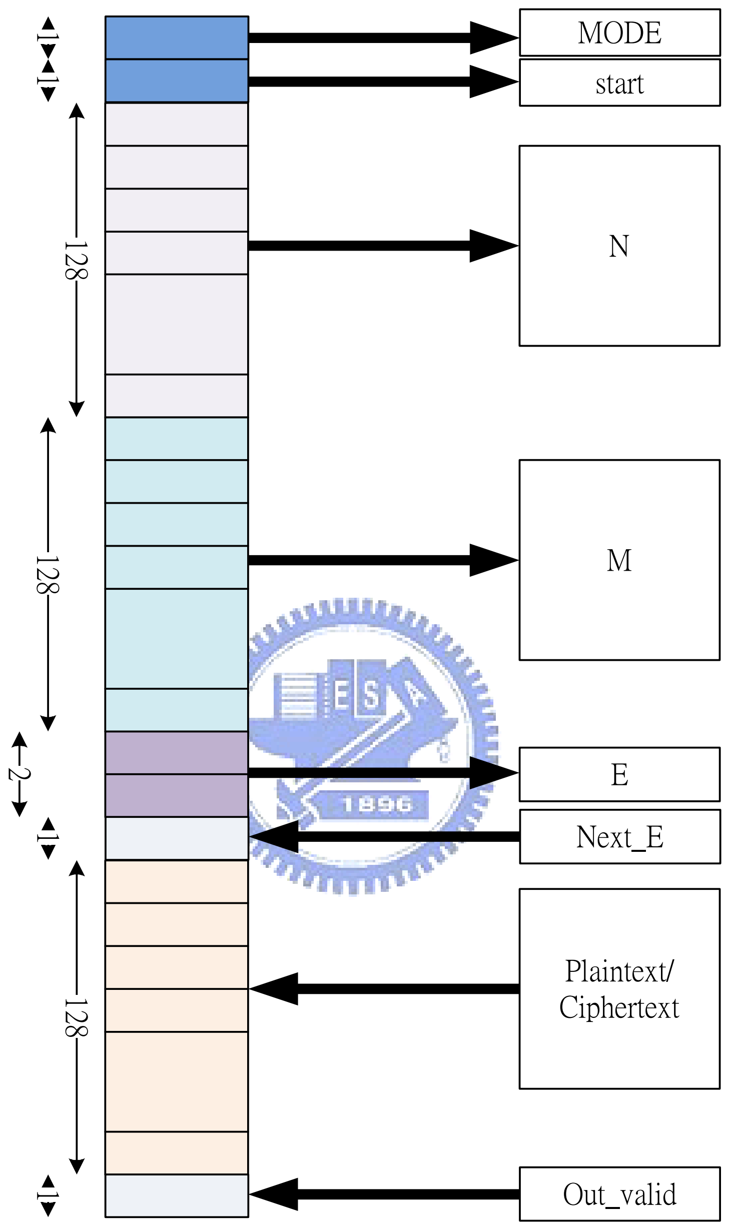


Figure B.4: AMBA address mapping.

Appendix C

Permutation Function

Table C.1 shows how the flexible output works. Recall that the input of Z_t in Figure 4.7 is the sum of different partial sum from PUs. The modulus length n is a 12-bit number which represent the number of bit being performed and $n^{'} = (n-1)\ mod64$. The output of $n^{'}$ is the sum of $(n^{'}+1)$th PU.

Table C.1: The permutation function of flexible output.

n'	output		n'	output	
0	$Z_{t-1}[0]$	, $Z_{t-2}[31:1]$	32	$Z_t[0]$	,$Z_{t-1}[31:1]$
1	$Z_{t-1}[1:0]$	, $Z_{t-2}[31:2]$	33	$Z_t[1:0]$	,$Z_{t-1}[31:2]$
2	$Z_{t-1}[2:0]$	, $Z_{t-2}[31:3]$	34	$Z_t[2:0]$	,$Z_{t-1}[31:3]$
3	$Z_{t-1}[3:0]$	, $Z_{t-2}[31:4]$	35	$Z_t[3:0]$	,$Z_{t-1}[31:4]$
4	$Z_{t-1}[4:0]$	, $Z_{t-2}[31:5]$	36	$Z_t[4:0]$	,$Z_{t-1}[31:5]$
5	$Z_{t-1}[5:0]$	, $Z_{t-2}[31:6]$	37	$Z_t[5:0]$	,$Z_{t-1}[31:6]$
6	$Z_{t-1}[6:0]$	, $Z_{t-2}[31:7]$	38	$Z_t[6:0]$	,$Z_{t-1}[31:7]$
7	$Z_{t-1}[7:0]$	, $Z_{t-2}[31:8]$	39	$Z_t[7:0]$	,$Z_{t-1}[31:8]$
8	$Z_{t-1}[8:0]$	, $Z_{t-2}[31:9]$	40	$Z_t[8:0]$	,$Z_{t-1}[31:9]$
9	$Z_{t-1}[9:0]$	, $Z_{t-2}[31:10]$	41	$Z_t[9:0]$	,$Z_{t-1}[31:10]$
10	$Z_{t-1}[10:0]$	, $Z_{t-2}[31:11]$	42	$Z_t[10:0]$	,$Z_{t-1}[31:11]$
11	$Z_{t-1}[11:0]$	, $Z_{t-2}[31:12]$	43	$Z_t[11:0]$	,$Z_{t-1}[31:12]$
12	$Z_{t-1}[12:0]$	, $Z_{t-2}[31:13]$	44	$Z_t[12:0]$	,$Z_{t-1}[31:13]$
13	$Z_{t-1}[13:0]$	, $Z_{t-2}[31:14]$	45	$Z_t[13:0]$	,$Z_{t-1}[31:14]$
14	$Z_{t-1}[14:0]$	, $Z_{t-2}[31:15]$	46	$Z_t[14:0]$	,$Z_{t-1}[31:15]$
15	$Z_{t-1}[15:0]$	, $Z_{t-2}[31:16]$	47	$Z_t[15:0]$	,$Z_{t-1}[31:16]$
16	$Z_{t-1}[16:0]$	, $Z_{t-2}[31:17]$	48	$Z_t[16:0]$	,$Z_{t-1}[31:17]$
17	$Z_{t-1}[17:0]$	, $Z_{t-2}[31:18]$	49	$Z_t[17:0]$	,$Z_{t-1}[31:18]$
18	$Z_{t-1}[18:0]$	, $Z_{t-2}[31:19]$	50	$Z_t[18:0]$	,$Z_{t-1}[31:19]$
19	$Z_{t-1}[19:0]$	, $Z_{t-2}[31:20]$	51	$Z_t[19:0]$	,$Z_{t-1}[31:20]$
20	$Z_{t-1}[20:0]$	, $Z_{t-2}[31:21]$	52	$Z_t[20:0]$	,$Z_{t-1}[31:21]$
21	$Z_{t-1}[21:0]$	, $Z_{t-2}[31:22]$	53	$Z_t[21:0]$	,$Z_{t-1}[31:22]$
22	$Z_{t-1}[22:0]$	, $Z_{t-2}[31:23]$	54	$Z_t[22:0]$	,$Z_{t-1}[31:23]$
23	$Z_{t-1}[23:0]$	, $Z_{t-2}[31:24]$	55	$Z_t[23:0]$	,$Z_{t-1}[31:24]$
24	$Z_{t-1}[24:0]$	, $Z_{t-2}[31:25]$	56	$Z_t[24:0]$	,$Z_{t-1}[31:25]$
25	$Z_{t-1}[25:0]$	, $Z_{t-2}[31:26]$	57	$Z_t[25:0]$	,$Z_{t-1}[31:26]$
26	$Z_{t-1}[26:0]$	, $Z_{t-2}[31:27]$	58	$Z_t[26:0]$	,$Z_{t-1}[31:27]$
27	$Z_{t-1}[27:0]$	, $Z_{t-2}[31:28]$	59	$Z_t[27:0]$	,$Z_{t-1}[31:28]$
28	$Z_{t-1}[28:0]$	, $Z_{t-2}[31:29]$	60	$Z_t[28:0]$	,$Z_{t-1}[31:29]$
29	$Z_{t-1}[29:0]$	, $Z_{t-2}[31:30]$	61	$Z_t[29:0]$	,$Z_{t-1}[31:30]$
30	$Z_{t-1}[30:0]$	, $Z_{t-2}[31]$	62	$Z_t[30:0]$	,$Z_{t-1}[31]$
31	$Z_{t-1}[31:0]$		63	$Z_t[31:0]$	

Bibliography

[1] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. IT-22, no. 6, pp. 644–654, 1976.

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

[3] T. E. Gamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," in *Proceedings of CRYPTO 84 on Advances in cryptology.* New York, NY, USA: Springer-Verlag New York, Inc., 1985, pp. 10–18.

[4] *PKCS#1: RSA Cryptography*, RSA Laboratories Std. 800-57, 2002.

[5] *Digital Signatures Using Reversible Public Key Cryptography for the Financial Services Industry - RSA digital signature technique*, ANSI Std. X9.31, 1998.

[6] E. Biham and A. Shamir, "Differential cryptanalysis of des-like crytptosystems," *Journal of Cryptography*, vol. 4, no. 1, pp. 3–72, 1991.

[7] M. Matsui, "Linear cryptanalysis method for des cipher," in *Proceedings of Advances in Cryptology-Eurocrypt '93.* Springer-Verlag, 1994, pp. 386–397.

[8] P. Kocher, "Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems," in *Proceedings of Advances in Cryptology-CRYPTO '96.* Springer-Verlag, 1996, pp. 104–113.

[9] P. Kocher, J. Jaffe, and B. Jun, "Introduction to differential power analysis and related attacks," in *http://www.cryptography.com/dpa/technical*, 1998.

[10] T. S. Messerges, E. A. Dabbish, and R. H. Sloan, “Power analysis attacks of modular exponentiation in smartcards,” in *Proceedings of Workshop on Cryptographic Hardware and Embedded Systems.* Springer-Verlag, August 1999, pp. 144–157.

[11] C. C. Yang, T. S. Chang, and C. W. Jen, “A new rsa cryptosystem hardware design based on montgomerys algorithm,” in *IEEE Trans. on Circuits and Systems - II: Analog and Digital Signal Processing*, vol. 45. New York, USA: IEEE Computer Society, July 1998, pp. 908–913.

[12] A. Daly and W. Marnane, “Efficient architectures for implementing Montgomery modular multiplication and RSA modular exponentiation on reconfigurable logic,” in *FPGA ’02: Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays.* New York, USA: ACM Press, 2002, pp. 40–49.

[13] T. Blum and C. Paar, “Montgomery modular exponentiation on reconfsaigurable hardware,” in *Proc. 14th IEEE Symp. Computer Arithmetic*, 1999, pp. 70–77.

[14] C. D. Walter, “Systolic modular multiplication,” *IEEE Transactions on Computers*, vol. 42, no. 3, pp. 376–378, Mar 1993.

[15] C.-H. Wang, C.-T. H. Chih-Pin Su and, and C.-W. Wu, “A word-based rsa cryptoprocessor with enhanced pipeline performance,” in *IEEE Asia-Pacific Conference on Advanced System Integrated Circuits.* New York, USA: IEEE Computer Society, 2004, pp. 218–221.

[16] L. A. Tawalbeh, A. F. Tenca, S. Park, and C. K. Koç, “Use of elliptic curves in cryptography,” in *Thirty-Eighth Asilomar Conference on Signals, Systems, and Computers*, vol. 1, November 2004, pp. 483–487.

[17] A. F. Tenca and Çetin Kaya Koç, “A scalable architecture for modular multiplication based on montgomery’s algorithm,” *IEEE Transactions on Computers*, vol. 52, no. 9, pp. 1215–1221, September 2003.

[18] D. Harris, R. Krishnamurthy, M. Anders, S. Mathew, and S. Hsu, “An improved unified scalable radix-2 montgomery multiplier,” in *ARITH’05: Proceedings of the*

17th IEEE Symposium on Computer Arithmetic. New York, USA: IEEE Computer Society, 2005, pp. 172–178.

[19] C. D. Walter, "Precise bounds for montgomery modular multiplication and some potentially insecure rsa moduli," in *Topics in Cryptology-CT-RSA 2002, B. Preneel (editor), Lecture Notes in Computer Science*, vol. 2271. San Jose, CA, USA: Springer Berlin / Heidelberg, 2002, pp. 30–39.

[20] Y.-L. Chen, "Design and implementation of reconfigurable rsa cryptosystems," Master's thesis, National Chiao Tung University, 2006.

[21] K. Mukaida, M. Takenaka, N. Torii, and S. Masui, "Design of high-speed and area-efficient montgomery modular multiplier for rsa algorithm," in *IEEE Symp. VLSI Circuits*, 2004, pp. 320–323.

[22] C. McIvor, M. McLoone, and J. V. McCanny, "Modified montgomery modular multiplication and rsa exponentiation techniques," in *IEE Proceedings Computers and Digital Techniques*, vol. 151, 2004, pp. 402–408.

[23] S. H. Tang, K. S. Tsui, , and P. H. W. Leong, "Modular exponentiation using parallel multipliers," in *Proceedings of the 2003 IEEE International Conference on Field Programmable Technology (FPT)*, 2003, pp. 52–59.

[24] *AMBA^{TM} Specification Rev 2.0*, ARM Ltd. Std., 1999.

作者簡介

姓名：林祐進

出生：彰化縣

學歷：台中市忠明國小、台中市向上國中、國立台中第一高級中學

91.9 ~ 95.6 國立交通大學電子工程學系

95.9 ~ 97.11 國立交通大學電子研究所系統組