針對安全性嵌入式系統之彈性管線化設計與實做

Design and Implementation of a Flexible Pipeline for Secure Embedded Systems

研 究 生：陳 治 瑋　　Student：Zhi-Wei Chen

指導教授：單 智 君　博士　　Advisor：Dr. Jean, Jyh-Juin Shann

國 立 交 通 大 學
資 訊 工 程 學 系
碩 士 論 文

A Thesis
Submitted to Department of Computer Science and Information Engineering
College of Electrical Engineering and Computer Science
National Chiao Tung University
in Partial Fulfillment of the Requirements
for the Degree of
Master
In

Computer Science and Information Engineering

June 2004

Hsinchu, Taiwan, Republic of China

中華民國 九十四 年 八 月

具安全考量之嵌入式系統的彈性管線化設計與實做

學生：陳治瑋　　　　　　　　　　　　　　　　　　　　指導教授：單智君 博士

國立交通大學資訊工程學系（研究所）碩士班

摘　要

在現今的環境中，提供加密的需求已是刻不容緩，如果在嵌入式系統中加入加密的運算，就會遇到幾項議題，其中我們針對處理速度以及硬體彈性這兩項議題進行討論，我們針對目前較常見的加密演算法，分別為 AES DES 和 RSA。提供一個可在AES，DES 和 RSA 之間彈性轉換，並且可以彌補速度上不足之硬體。在考量處理速度不足這項議題之下，我們採用速度與面積乘積為評比標準。

在本論文中，我們首先分析此三演算法之運算需求，然後針對不同類型之運算分別設計出排列組合單元，運算單元以及記憶單元，其中排列組合單元採客制化設計，運算單元由處理單元所組成，記憶單元則由單位緩衝區所組成，我們討論處理單元以及單位緩衝區的設計以及考量在不同比例之運算單元以及單位緩衝區之下，造成面積速度乘積的影響，最後所提出的設計和針對個別演算法之客制化設計做比較，比較結果顯示我們的方法確實在面積速度乘積有較好的效果。

Design and Implementation of a Flexible Pipeline for Secure Embedded Systems

Student：ZhiWei Chen Advisor：Dr, Jean, J.J Shann

Institute of Computer Science and Information Engineering
National Chiao-Tung University

Abstract

Providing security has become more and more urgent and necessary in embedded systems. If we want to support security in our embedded systems, some issues must be solved. We focus on processing gap and flexibility concerns. We target on the three commonly used cryptographic algorithms, AES, DES and RSA. In our thesis, we want to propose a hardware which solves the processing gap and switches flexibly between AES, DES, and RSA. Under the consideration of processing gap, we use space-time product as our performance metrics.

We first classify the operation of the three cryptographic algorithms into three classes. Then, we design modules for different operation classes respectively. The three modules are permutation-combination unit, computation unit and memory unit. The permutation-combination unit is a custom design. The computation unit is consisted of processing elements and the memory unit is consisted of tile buffers. The different ratio of processing elements and tile buffers will lead to different results. We choose the most appropriate ratio. Finally, our proposed method will get better result than ASIC design.

誌　謝

首先感謝我的指導老師 單智君教授，在老師的諄諄教誨、辛勤指導與勉勵下，我得以順利完成此論文，並且順利通過畢業口試。同時感謝我的口試委員 鍾崇斌教授以及謝萬雲教授，由於他們的指導與建議，讓這篇論文更加完整和確實。

此外，感謝實驗室的學長—蔣昆成學長和喬偉豪學長，每次都不厭其煩地跟我討論許多問題，還帶我去吃海鮮，給予我莫大的幫助。也感謝實驗室全體學長姐、同學以及學弟們，真的很高興可以認識你們大家。因為你們，讓我的研究生活充滿了歡樂跟 dota。

最後感謝我的家人，謝謝你們在背後全心全意地支持我，讓我在這研究的路上走得更順利，進而能無後顧之憂的學習，讓我追求自己的理想。

謹向所有支持我、勉勵我的師長與親友，奉上最誠摯的祝福，謝謝你們。

陳治瑋

2005. 8. 15

Table of contents

List of Figures

List of Tables

國立交通大學

資訊工程系

碩 士 論 文

針對安全性嵌入式系統之彈性管線化設計與實做

Design and Implementation of a Flexible Pipeline for Secure Embedded Systems

研 究 生：陳 治 瑋

指導教授：單 智 君 博士

中 華 民 國 九 十 四 年 八 月

Chapter 1 Introduction

Curiosity about other people's business and the hiding of information are characteristic of all human societies. In military, commerce, and diplomat, protecting some confidential papers is more and more important. Furthermore, embed security feature in some personal devices, such as PDA and handset, has become a considerable factor for consumers. According to the statistic of Mobile Commerce taken by ***ePaynews.com,*** nearly 52% of cell phone users and 47% of PDA users feel that security is the largest concern preventing the adoption of Mobile Commerce. The statistic bar graph is shown in Fig.1-1. In the Fig. 1-1, the second important reason is the klunky user experience. It means the habits of experienced users. The two reasons account for 70% of the statistics. [1]

Besides the urgent necessity, the stronger computing power and the more impeccable development of mathematical techniques have make cryptosystems extremely sophisticated. It's able to construct ciphers that are effective and impossible to break. As a result, embedded systems adapt some security features is necessary and possible.

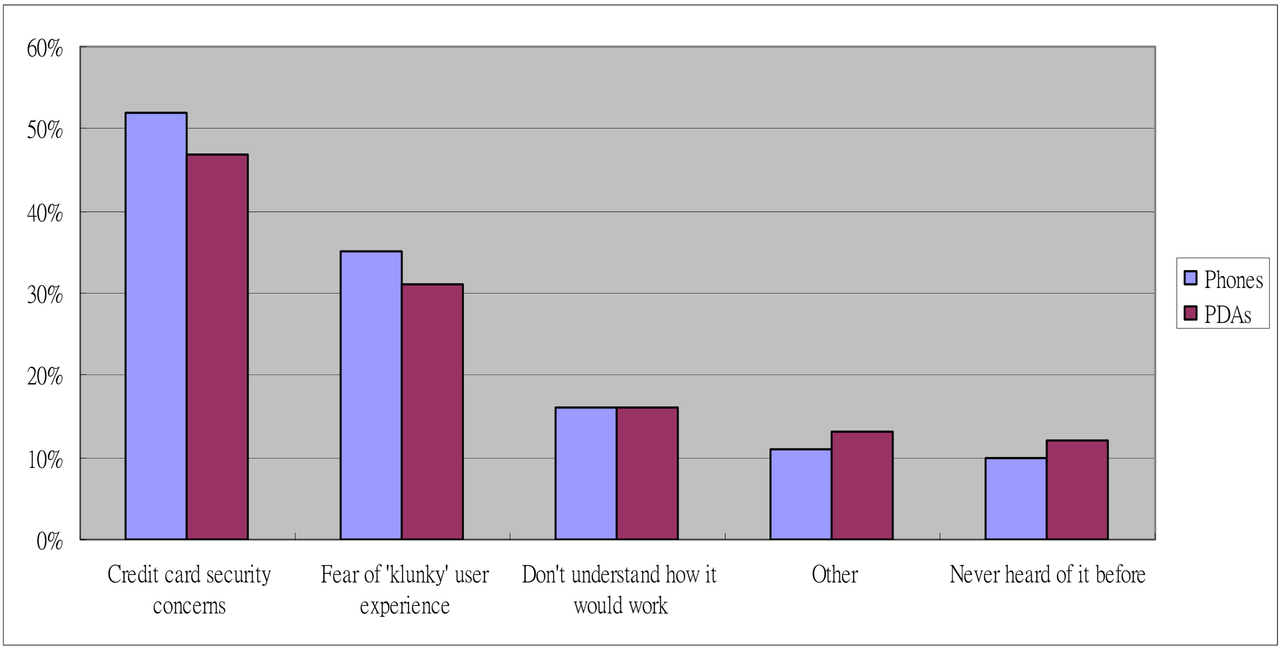


Fig. 1-1 Obstacles preventing consumers from adopting Mobile Commerce [1]

1.1 Design Challenges of Secure Embedded Systems

Many embedded systems are constrained by the environments they operate in, and by the resources they possess. For secure embedded systems, there are some new design challenges. These new design challenges are processing gap, flexibility concerns, battery gap, tamper resistance, assurance gap and cost. The processing gap means the gap between bandwidth of devices and environments. The flexibility concerns come from the characteristics of cryptographic algorithms. Battery gap is the insufficiency of battery. The tamper resistance is the countermeasure for malicious software such as viruses and Trojan horses. The assurance gap is the gap between current systems and the reliable systems which never crash under any kind of situation. Last but not least, the cost is the area constrain of embedded environment. [2] Our thesis is focus on processing gap and flexibility concerns.

1.1.1 Processing Gap

At different environments, data rates of cellular (128 kbps~2Mbps), wireless Lan (2~60Mbps) and the lower-end of access network (~100Mbps) are supported by current low- and high-end embedded processors. [2] Take XScale as an example, when it 100% dedicates for record protocol of Secure Socket Layer (SSL) which is a popular security protocol, it can only sustain data rate 3.1Mbps. Any higher rates are unattainable. If we use this embedded process for SSL, this would lead processing gap. In Fig.1-2 shows the throughput of low- and high-end processors. The horizontal axis is MIPS of embedded processors. The vertical axis is data rate produced by these embedded processors. We choose some popular processor. For low-end system, we choose the XScale, ARM9 and SA-1110 as an example. For high-end system, we choose Xcon and PIII.

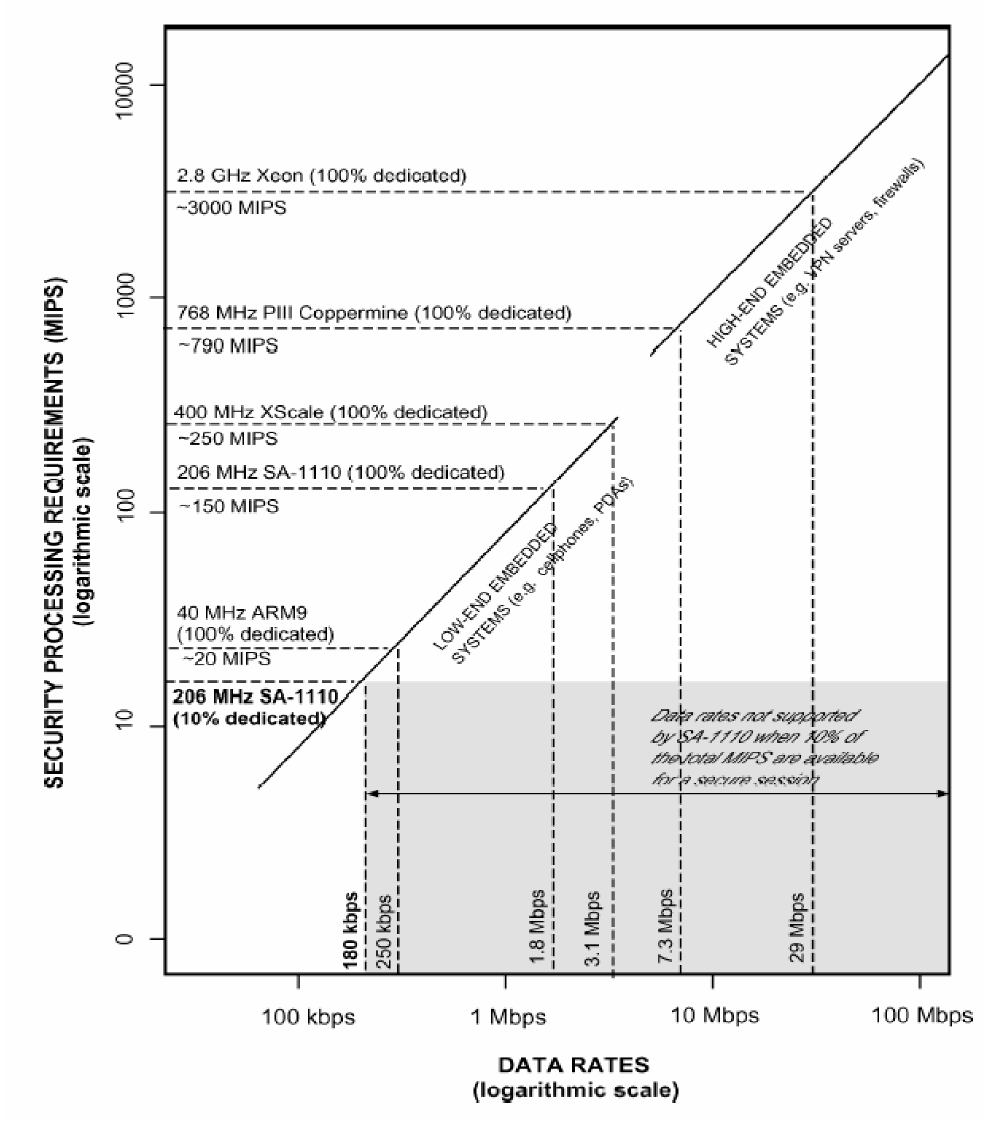


Fig. 1-2 Throughput of high- and low- end embedded processors [2]

1.1.2 Flexibility Concerns

A typical security protocol standard usually allows for a wide range of cryptographic algorithms. In general, asymmetric cryptographic algorithms are used to exchange the keys which needed by symmetric ones. If we use ASIC to speed up the two cryptographic algorithms individually, hardware utilization will be very low. The low utilization comes from using symmetric and asymmetric ASIC in sequence. When we use asymmetric cryptographic algorithms to exchange keys, the hardware used to speed up symmetric cryptographic algorithms is idle. It is the same for asymmetric ones when symmetric ones are active.

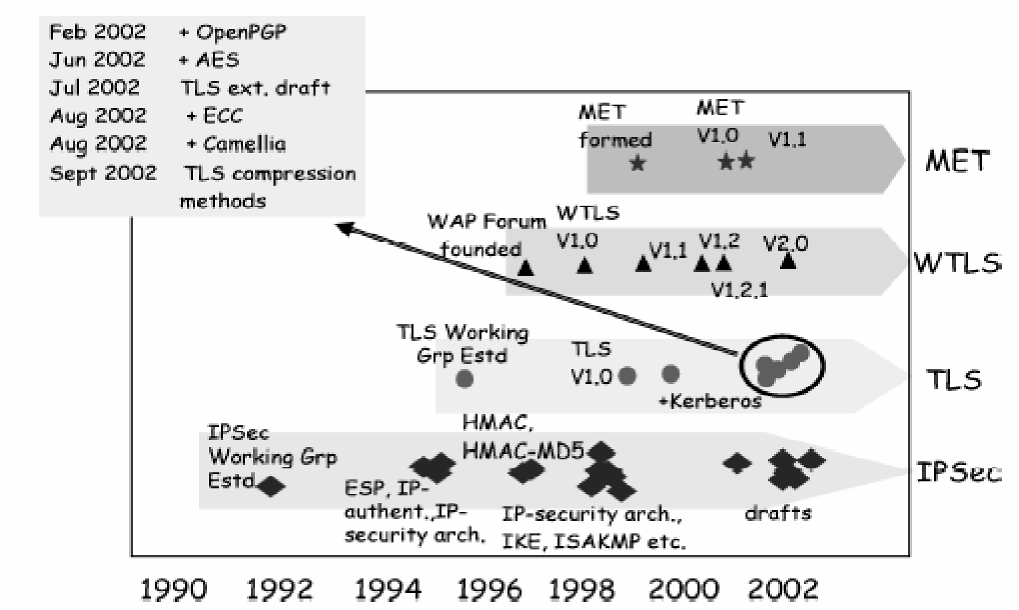


Fig. 1-3 Evolution of security protocols [2]

Besides the requirement of supporting different cryptographic algorithms, security protocols and cryptographic algorithms are not only diverse, but also continuously evolving over time. As time goes by, future computation power will become stronger than current one. The current cryptosystem will become insecurity. So, we need some new cryptographic algorithms or some cryptographic enhancements. As the Fig.1-3 shows, these security protocols add more and more features from 1990 to 2002. For cryptographic algorithms itself, take DES as an example. 3DES is an enhancement algorithm with respect to DES. It uses the same cryptographic algorithm, but need longer key to support higher security level.

As a result, embedded systems supporting for security need some kinds of flexibility concerns between different cryptographic algorithms and forward compatible for future ones.

1.2 Architecture for Security Processing

As the security embedded systems grow, there are two generations of security processing architecture. The First-generation solutions perform security processing by executing security software on the embedded processors. Because they use software to perform cryptographic algorithms, they have high flexibility and fast turn around time. Unfortunately, the characteristic of processor which is designed to execute any kind of application leads doesn't meet the case for stream data processing. The most execution time is spent on instruction fetch and decode. Nevertheless, the applications such like cryptographic algorithms are fixed. As a result, the First-generation solutions are not efficient in terms of their performance and energy consumption.

The First-generation solutions have the defect of performance. Some people suggest that design a dedicated hardware to speed up cryptographic algorithms. It is a good approach in the view point of processing gap. So, the Second-generation solutions are proposed. The Second-generation ones sacrifice the flexibility and turn around time. The benefits of them are hardware efficiency and low power. These advantages are due to the custom design. But, as the section 1.1 tells us, the cryptographic algorithms are not only diverse, but also continuously evolving over time. As time goes by, the mainstream cryptographic algorithms may be replaced by some new cryptographic ones. Under this situation, the Second-generation solutions will get troubles. As a result, we need the third-generation solutions which need to have benefits of first- and second-generation ones. They need to have high efficiency, high flexibility and fast turn around time. Fig.1-4 shows the pro and con of the three generations of security processing architecture.

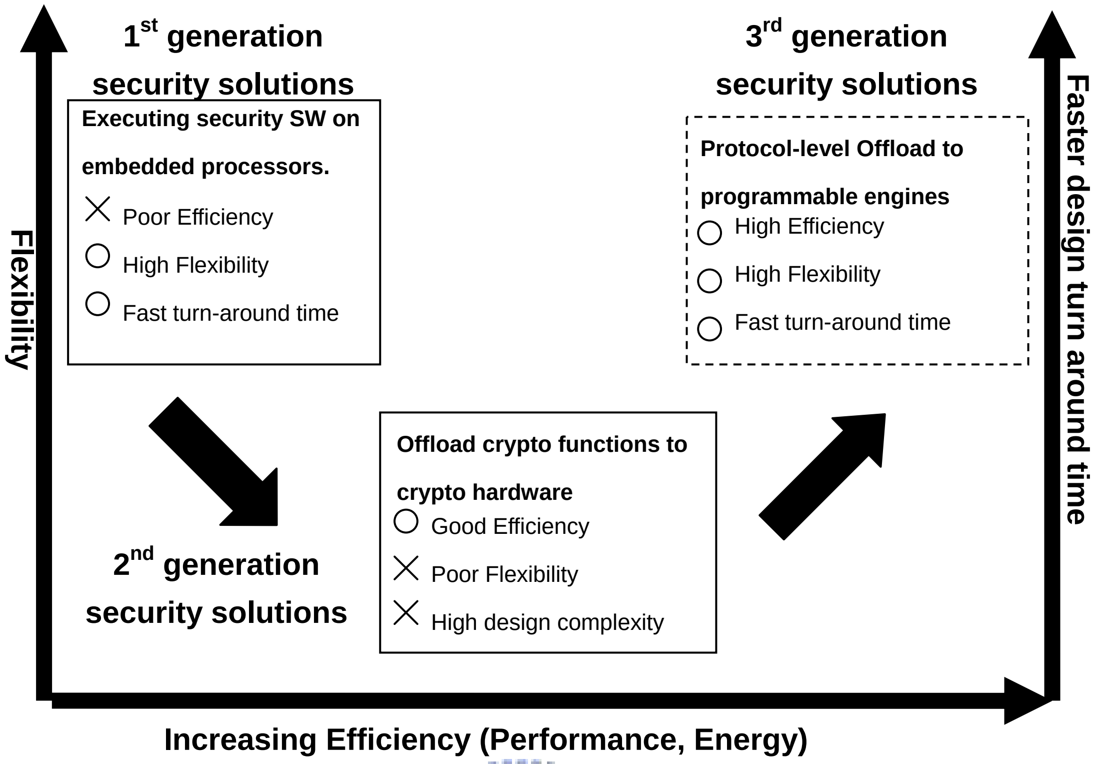


Fig. 1-4 Security processing architectures [2]

1.3 Motivation and Objective

For secure embedded systems, processing gap still needs to be solved. But new issue for embedded systems, flexibility, should be considered. Unfortunately, the researches about flexibility are rare. In our thesis, we propose a hardware which solves the processing gap and switches flexibly between AES, DES, and RSA. Under consideration of processing gap, we use space-time product as our performance metrics.

We first classify the operation of the three cryptographic algorithms into three classes. Then, we design modules for different operation classes respectively. The three modules are permutation-combination unit, computation unit and memory unit. The permutation-combination unit is a custom design. The computation unit is consisted of processing elements and the memory unit is consisted of tile buffers. The different ratio of processing elements and tile buffers will lead to different result. We choose the most appropriate ratio. Finally, our proposed method will get better result than dedicated design. Organization of this thesis is that chapter 2 will introduce some cryptographic concepts, the three algorithms, DES, AES, and RSA and some hardware implementations. Chapter 3 is our proposed design. Chapter 4 is evaluation results. We compare our design to dedicated design. Chpater 5 is conclusion and future work.

Chapter 2 Background and Related Work

In this chapter, we will give an overview of cryptography. Then, we introduce the three commonly used cryptographic algorithms and summarize all the used operation. Finally, previous works related of the three cryptographic algorithms are presented.

2.1 Overview of Cryptography

The word **Cryptography** is composed by two ancient Greek words. “kryptŏs” and “graphein” The “kryptŏs” means hidden and “graphein” means writing. **Cryptography** is the study of information hiding, message certification and the science of encrypting and decrypting text. [3][4]

Cryptography has existed thousands of years. The Ancient Greece, Spartan, wraps leather at a specific ruler and then write the information on it to transmit to others; Others which want to read the message only need the ruler with equal size. When they get the leather like this, they wrapped the leather on the ruler and then read the original information. In this way, even if this leather is intercepted midway. It is only some useless information in a mess because of unknown ruler’s length. This is one of the earliest cryptography of the mankind that records in history. Of course, such system is ludicrously weak. The modern cryptosystems use sophisticated algorithms based on mathematical problems that are difficult to solve.[5]

In general, a cryptosystem will have three roles during messages exchanging. They are sender, receiver and intruder. Typical cryptosystems are shown as Fig.2-1. At the sender side, sender encrypts plaintext M with an encryptor E and a key k_1. The ciphertext is $C = E(M, k_1)$. Then, the sender sends the ciphertext to public channel for receiver. When receiver receives the ciphertext, he uses a decryptor D and key a k_2 to decrypt ciphertext to plaintext. $M = D(C, k_2) = D(E(M, k_1), k_2)$. The intruder is a malicious role. He listens to the public channel and uses any kind of methods to know the plaintext from sender or pretend message to receiver. He can get ciphertext, but has no idea about $\boldsymbol{k_2}$.[4]

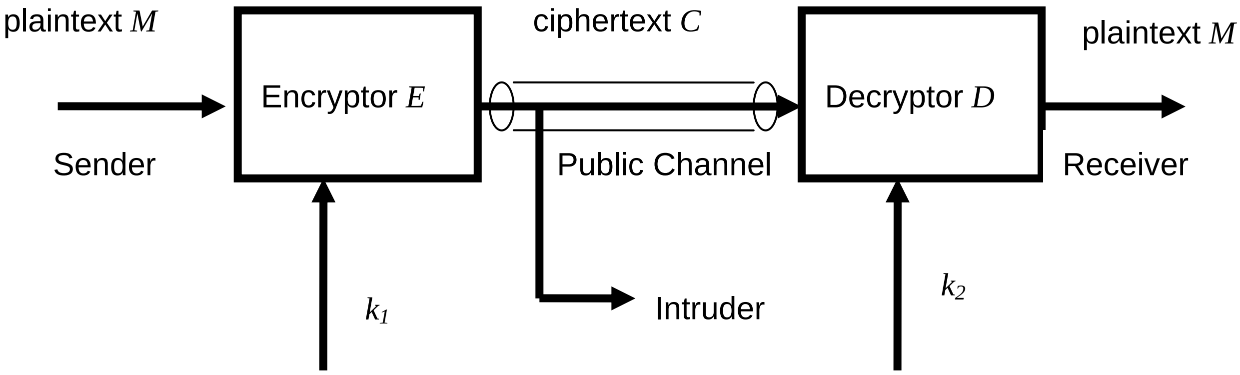


Fig. 2-1 Typical cripto-algorithm block diagram [4]

2.2 Symmetric and Asymmetric Cryptosystems

In typical cryptosystem, if secret $\boldsymbol{k_1}$ and $\boldsymbol{k_2}$ are the same, this cryptosystems are **Symmetric Key Cryptosystem, One-key Cryptosystem or Private Key Cryptosystem**. **Symmetric Key cryptosystems** have been used for thousands of years. They range from simple substitution ciphers to more complex constructions. One of the simplest form is known as the **Caesar cipher** used by *Julius Caesar*. The process of **Caesar cipher** is simply shifting the alphabet [6]. This system is very easy to break. Fortunately, the growth of computing power and some new developments in mathematics make that it is possible to create **Symmetric Key Cryptosystems** that are unbreakable. **Symmetric Key Cryptosystems** are generally very fast. But they have a disadvantage. Sender and receiver need to agree on the shared key previously. However, the communicating parties may never meet over the network. It is impossible for the two parties to encrypt data without having a shared secret key that is known in advance. So, the **Symmetric Key Cryptosystems** are vulnerable.

The secret key sharing can be a major vulnerability in **Symmetric Key**

Cryptosystems. In 1976, Diffie and Hellman[7] demonstrated an algorithm which is known as Diffie-Hellman Key Exchange. It is an elegant approach toward secure communication that has led to the development of **Public Key Cryptosystem**, also named **Asymmetric Cryptosystem**. The use of **Public Key Cryptosystem** is quite simple. As shown in Fig.2-2., sender and receiver share a single public key, but receiver has one more key, private key. The public key is available to everyone in the world including the intruder, but the private key is only known by receiver. If sender wants to send message to receiver, sender uses the public key to encrypt message and then send to receiver. When receiver receives the ciphertext, he uses his private key to decrypt message. As a result, sender and receiver don't have to exchange key previously. So, the **Asymmetric Cryptosystems** don't have the vulnerability of **symmetric cryptosystems**.

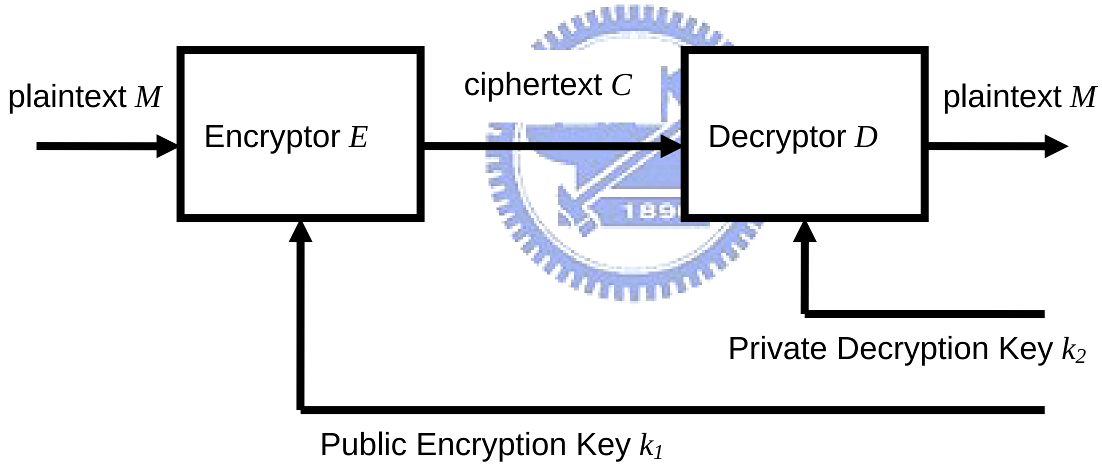


Fig. 2-2 Public Key Cryptosystem

Symmetric cryptosystems are faster than asymmetric ones. They are the preferred mechanism for encrypting large amount of message. A cipher such as DES[8] will be at least 100 times faster than the asymmetric cipher RSA[9] in software and might be up to 10,000 times faster when implemented on specialist hardware.[6] **Asymmetric cryptosystems** are most suitable for protecting data with high security requirement. In practice, the most satisfactory methods are combining both symmetric and asymmetric systems. Use asymmetric systems to exchange secret key which is used by symmetric ones.

After secret key exchanging, the symmetric cryptosystems can encrypt or decrypt data with this key. DES and AES are the commonly used symmetric algorithms, and RSA is asymmetric one. In our research, we will focus on the three cryptographic algorithms.

2.3 AES[11]

In October 2000, the NIST chose Rijndael as the new Advanced Encryption Standard (AES). AES is intended to replace DES and Triple DES as a new secure standard [10]. AES is a symmetric block cipher. It can process block data of 128-bit. The Fig.2-3 shows the AES encryption flow. Just like all symmetric cryptographic algorithms. The AES has a regular computation flow. It just repeats the routine, **round**, some times depending on different length of cipher key. The length of cipher key are 128-, 192- and 256-bit respectively. The AES encryption and decryption are composed by five main components. All of them are **Key Expansion, Subbytes, ShfitRows, Mixcolumn, and AddRoundKey**. [11]

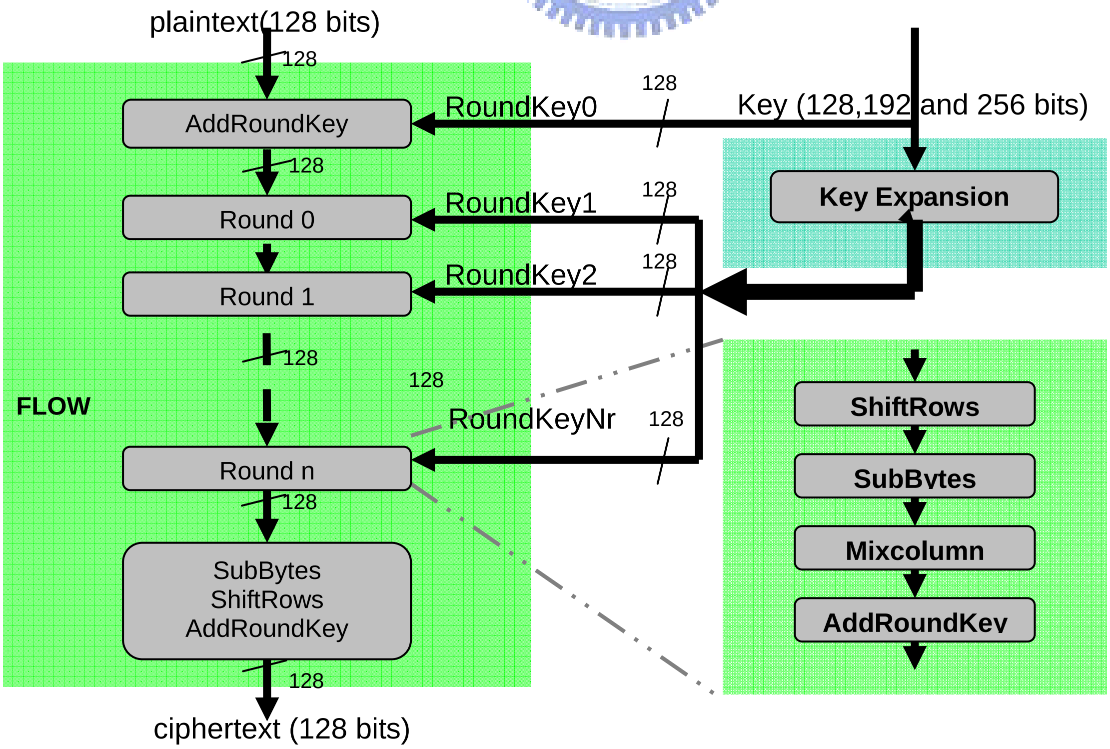


Fig. 2-3 Flow of AES

2.3.1 Definition

Before introducing AES, we must define some terms previously. The length of input, output, and **state** which is output of round in AES is 128 bits. This is represented by ***Nb*** = 4, which means the number of 32-bit words. The length of the cipher key, *k* is 128-, 192-, or 256-bit. The length of key is represented by ***Nk***. The 128, 192, and 256 are ***Nk*** = 4, 6, and 8 respectively. Depending on different length of cipher key, AES algorithm performs some times of round. The round number is represented by ***Nr***. When ***Nk*** is 4, 6, and 8, it means 10, 12 and 14 rounds in AES respectively. The combinations of Key-Block-Round are given as follow in Table. 2-1

	Cipher Key length(*Nk*)	**Block Size(*Nb*)**	**Number of Rounds(*Nr*)**
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

Table 2-1 Key-Block-Round Combination

2.3.2 Key Expansion

The **Key Expansion** algorithm takes cipher key to produces a key schedule for en/decryption flow. The Key Expansion will generate total ***Nb(Nr + 1)*** words. The **Key Expansion** processes shows as the following algorithm.

The SubBytes() is a function that perform four bytes table look up which will mention in section 2.3.3. The function RotWord() take a word [A1,A2,A3,A4] to perform a cyclic rotate, and then return the result [A2,A3,A4,A1]. And the Rcon[i] are constant in the form [X_i,0,0,0]. X_i are list as following table 2-2.

Round i	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
X_i	01	02	04	08	10	20	40	80	1b	36

Table 2-2 X_i of Rcon[i]

```
KeyExpansion(byte key[4×Nk], word w[Nb×(Nr+1)], Nk)
begin
    word temp
    i = 0
    while (i < Nk)
        w[i] = word(key[4×i], key[4×i+1], key[4×i+2], key[4×i+3])
        i = i+1
    end while
    i = Nk
    while (i < Nb × (Nr+1)]
        temp = w[i-1]
        if (i mod Nk = 0)
            temp = SubBytes(RotWord(temp)) xor Rcon[i/Nk]
        else if (Nk > 6 and i mod Nk = 4)
            temp = SubWord(temp)
        end if
        w[i] = w[i-Nk] xor temp
        i = i + 1
    end while
end
```

Take *Nk* = 4 as an example, the expansion executes as following Fig. 2-4

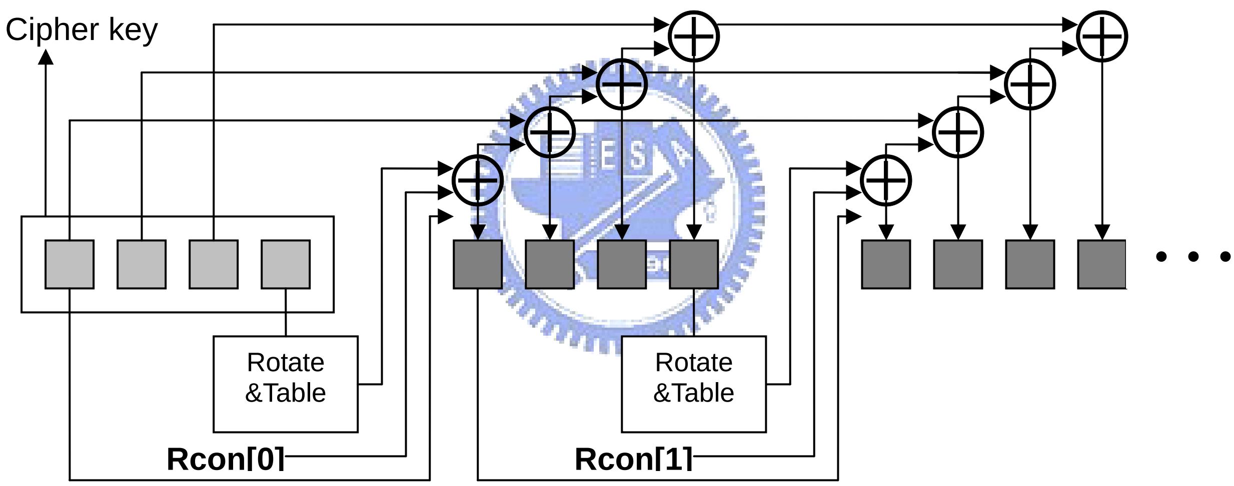


Fig. 2-4 Key Expansion Flow

2.3.3 SubBytes

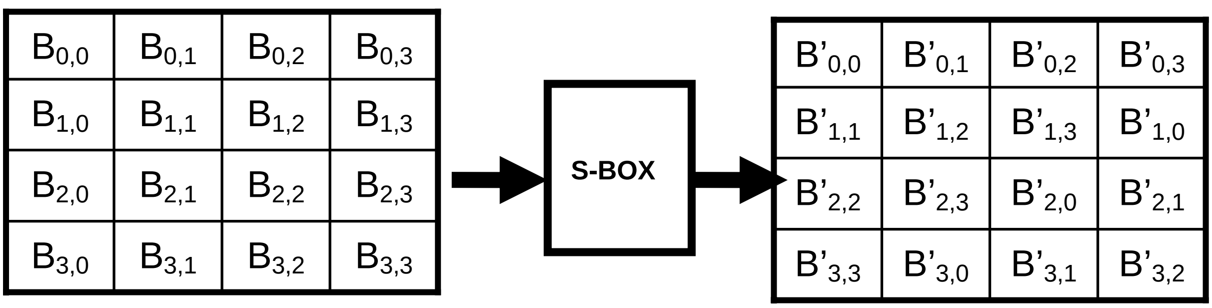


Fig. 2-5 SubBytes applies S-box to each byte of the state

The SubBytes transformation performs a non-linear byte substitution that operates independently on each byte as in Fig. 2-5. The non-linear byte substitution is constructed by the following transformation in Fig. 2-6. This transformation is invertible.

$$\begin{bmatrix} b_0' \\ b_1' \\ b_2' \\ b_3' \\ b_4' \\ b_5' \\ b_6' \\ b_7' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Fig. 2-6 SubBytes transformation

2.3.4 ShiftRows

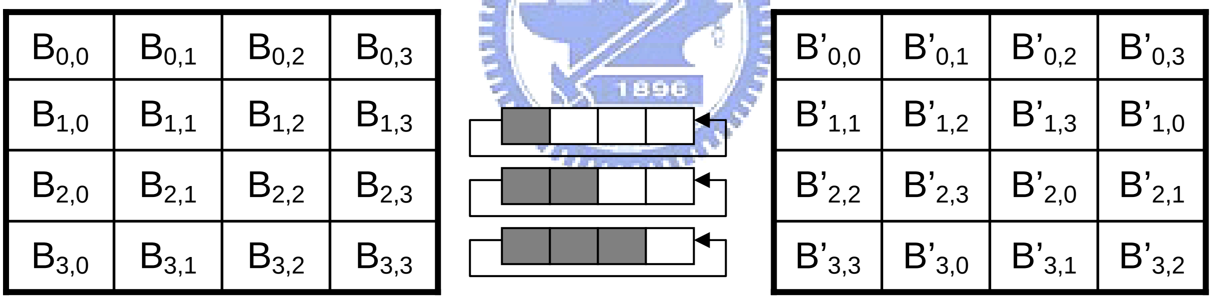


Fig. 2-7 shift cyclically the state

In the ShiftRows transformation, the last three rows of the **state** rotate over different numbers of bytes like Fig.2-7. The shift numbers are listed in table 2-3.

No. of column	C1	C2	C3
Left shift (Encryption)	1	2	3
Right shift (Decryption)	1	2	3

Table 2-3 Shift number C1 means Column one

2.3.5 MixColumn

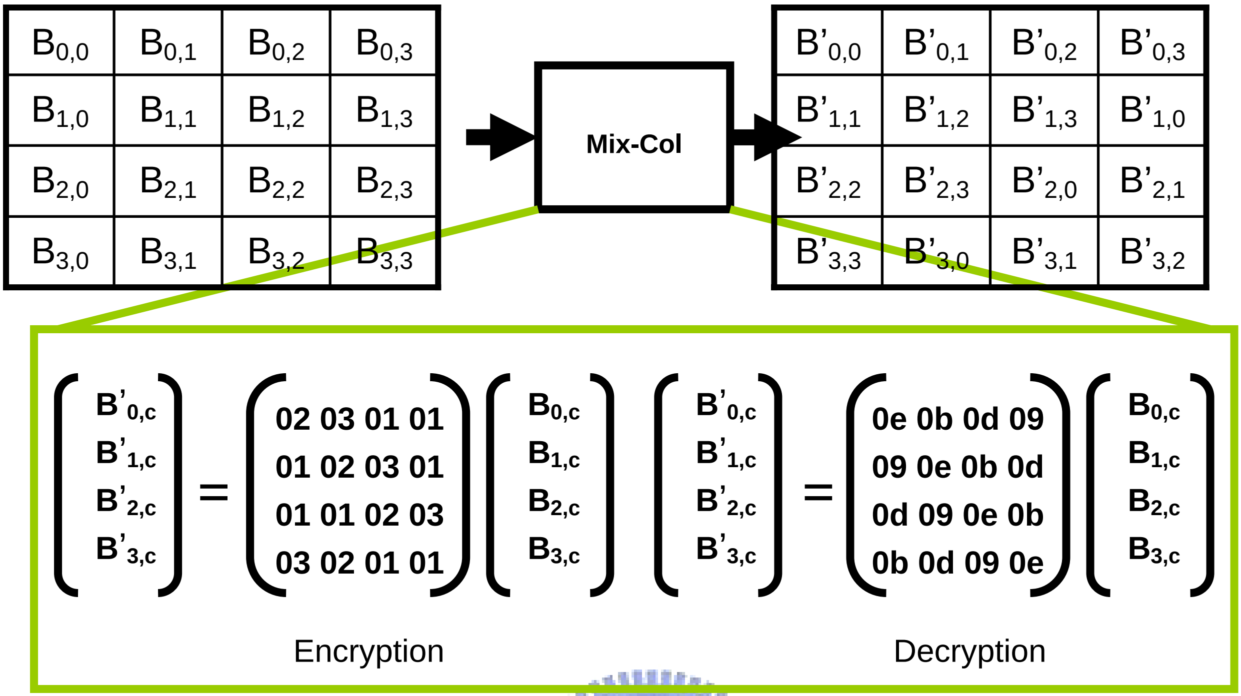


Fig. 2-8 Mixcolumn

The MixColumn operates column by column on the **state**. We can view it as a special matrix computation. It is shown as Fig.2-8. Before introducing the MixColumn, we must define some terms.

Definition

- $B_{i,C}$ ***is byte*** : $B_{i,C} = \{b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0\}$
- $B_{0,C} \oplus B_{1,C}$: $B_{0,C}$ ***XOR*** $B_{1,C}$
- $01 \otimes B_{0,C}$: $\{b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0\}$
- $02 \otimes B_{0,C}$: $\{b_6, b_5, b_4, b_3 \oplus b_7, b_2 \oplus b_7, b_1, b_0 \oplus b_7, b_7\}$
- ***Hardware form of "$02 \otimes B_{0,C}$" is shown as follows It is named X_TIME***

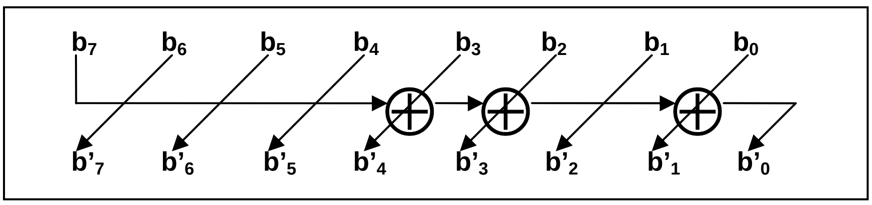


Fig. 2-9 X_TIME

Use the previous define, the encryption matrix can be expanded as follows.

$$B'_{0,C} = 02 \otimes B_{0,c} \oplus 03 \otimes B_{1,c} \oplus 01 \otimes B_{2,c} \oplus 01 \otimes B_{3,c}$$

$$= 02 \otimes B_{0,c} \oplus 02 \otimes B_{1,c} \oplus 01 \otimes B_{1,c} \oplus 01 \otimes B_{2,c} \oplus 01 \otimes B_{3,c}$$

The decryption shows as follows.

$$B'_{0,C} = 0e \otimes B_{0,c} \oplus 0b \otimes B_{1,c} \oplus 0d \otimes B_{2,c} \oplus 09 \otimes B_{3,c}$$

$$= 02 \otimes B_{0,c} \oplus 02 \otimes (02 \otimes B_{0,c}) \oplus 02 \otimes (02 \otimes (02 \otimes B_{0,c})) \oplus$$

$$01 \otimes B_{1,c} \oplus 02 \otimes B_{1,c} \oplus 02 \otimes (02 \otimes (02 \otimes B_{1,c})) \oplus$$

$$01 \otimes B_{2,c} \oplus 02 \otimes (02 \otimes B_{2,c}) \oplus 02 \otimes (02 \otimes (02 \otimes B_{2,c})) \oplus$$

$$01 \otimes B_{3,c} \oplus 02 \otimes (02 \otimes (02 \otimes B_{3,c}))$$

The Data flow graph of the two functions is shown as follows.

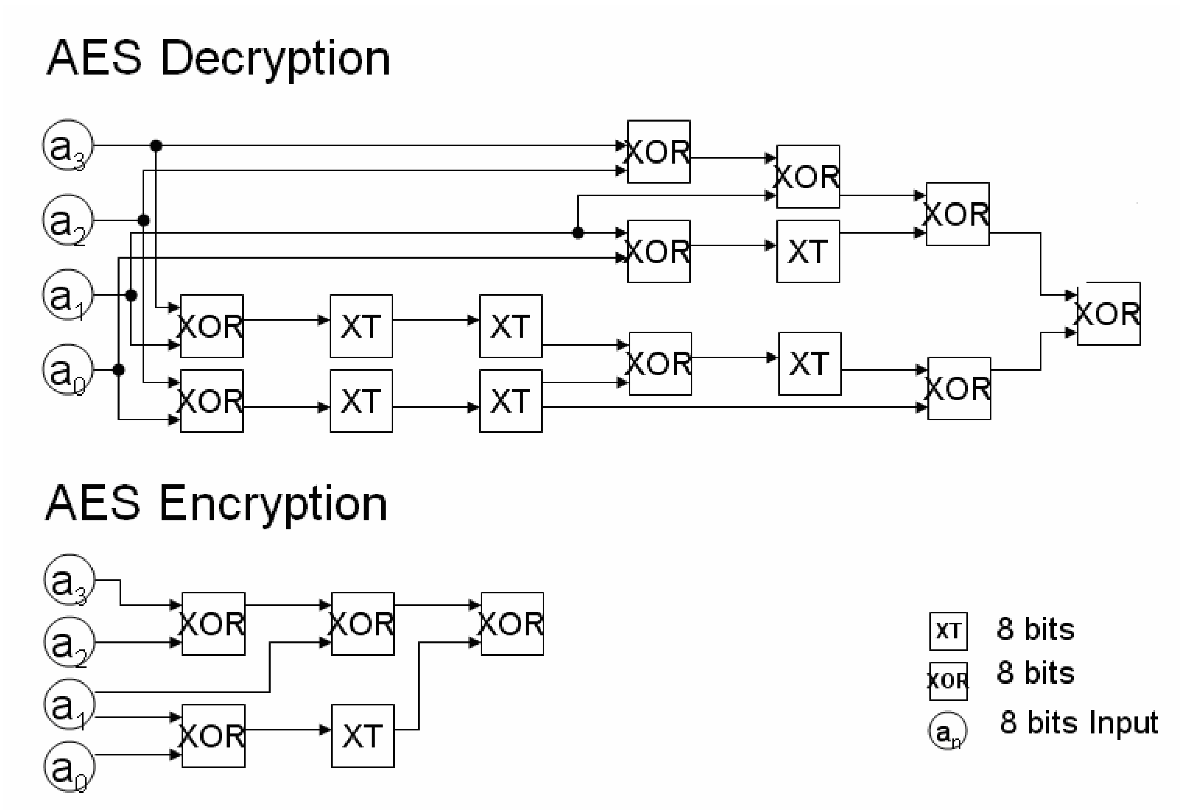


Fig. 2-10 Data flow graph of encryption and decryption

2.3.6 Add Round Key

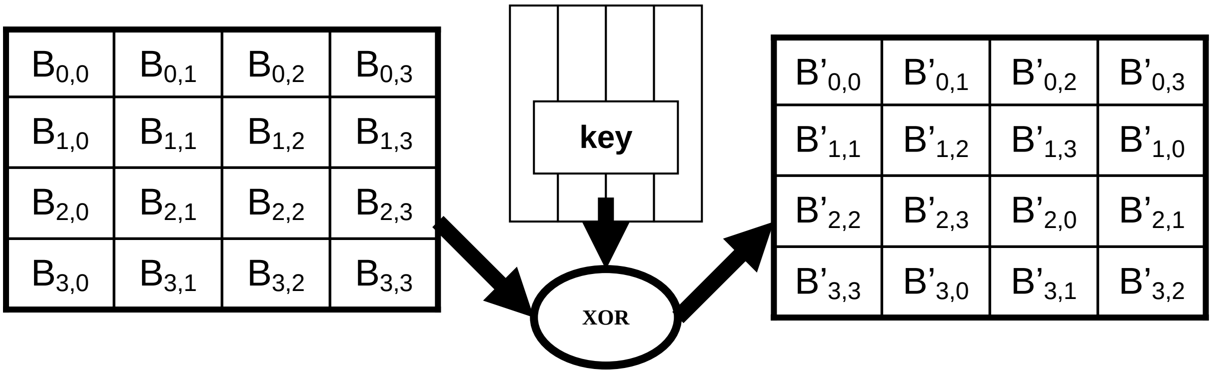


Fig. 2-11 Add Round Key

In this transformation, it adds **state** and key scheduling like Fig.2-11

2.4 DES[8]

Data Encryption Standard (DES) derives from work done by IBM. It became effective in July 1977 and reaffirmed in 1983, 1988 and 1999. It is probably the most widely used secret key cryptosystem, particularly in securing financial data, such as Automated Teller Machines (ATMs).

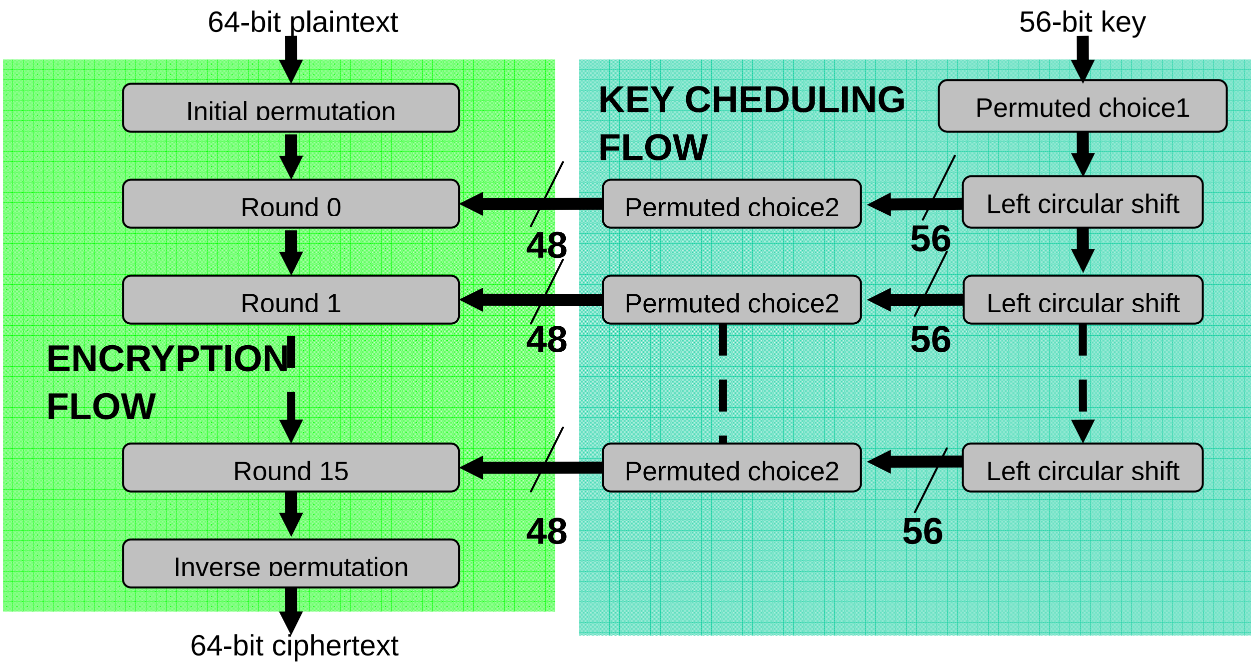


Fig. 2-12 Flow of DES

The DES flow is in Fig.2-12. It is composed by sixteen rounds and two permutations. The two permutations are place at beginning and end of DES round. In the key scheduling, DES needs only three kinds of permutations. As a result, we will divide DES in to two categories, **Permutations** and **Round**.

2.4.1 Permutations

In the Shift Left transformation, it treats input as two 28-bit blocks. Let's note the 2 28-bit as C and D. The left rotating one bit means a rotation of the bits one place to the left. So after rotating one bit C and D become {C[0],C[28]…..C[1]}, {D[0],D[28]…..D[1]} separately. The rotate number depends on round iteration number. It shows as the following table.

Iteration	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
rotate bits	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Table 2-4 Left shifts number

The Initial permutation, ***IP*** encipher the input block to the following permutation

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

That is the permuted input has bit 58 of the input as its first bit, bit 50 as its second bit, and so on with bit 7 as its last bit. The inverse permutation, IP^{-1}, is list as following by the same notation as ***IP***:

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

Key scheduling is performed by three permutations Permutation choice 1 **(PC1)**, Permutation choice 2 **(PC2)** and some left circular shift. **PC1** is determined as follows.

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

PC2 is determined as follows.

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

After Cipher Key performs permutation choice 2, it becomes 64 bits to 56 bits.

2.4.2 Round

DES is composed by sixteen rounds. The Round organization is shown as follows in Fig.2-13. We first divide the input 64-bit into **R**ight part and **L**eft part. The **R** part is the **f** input. We XOR the result of **f** with **L** part to be **R** part of the next round. And the current **R** part will be the **L** part of the next round.

The function **f** performs the following operation. First, **f** expands input from 32-bit to 48-bit, and then XOR with Key scheduling. Second, separate the 48-bit into eight 6-bit data and then the eight 6-bit will look up 8 different tables individually. Each table will output 4-bit data. Finally, the eight 4-bit data will perform the finial permutation. The **f** finishes.

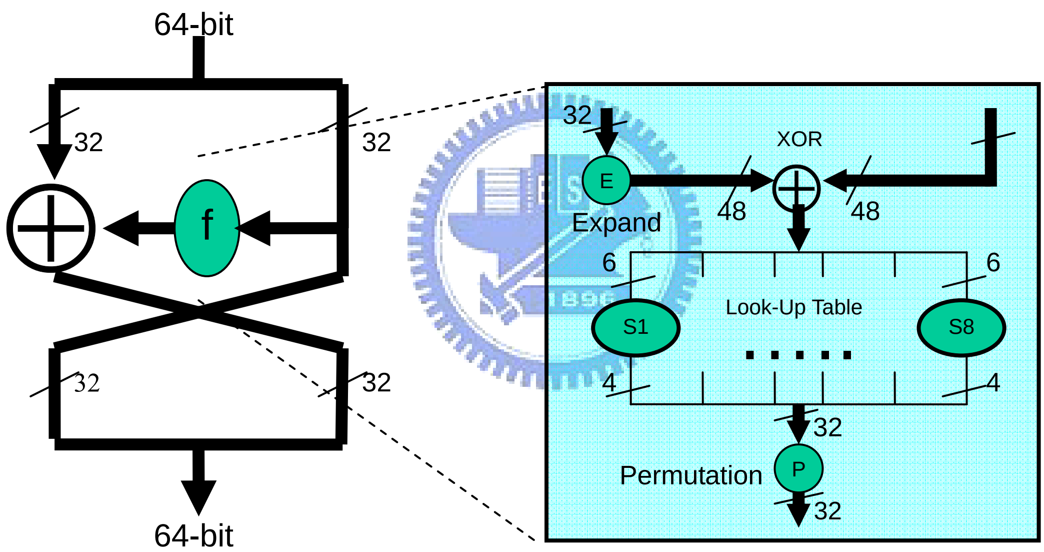


Fig. 2-13 Organization of Round

The function of **E**

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

The function of **P**

16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

2.5 RSA[9]

In 1976, Diffie and Hellman propose an algorithm which leads to development of today's **Public Key Cryptosystems**. After one year, three researchers at MIT used the suggestion of Diffie and Hellman to develop a method. The Three MIT researchers are Ron Rivest, Adi Shamir, and Leonard Adleman. The method is named after its founders, RSA. The RSA algorithm has become almost synonymous with **Public Key Cryptosystems**.

2.5.1 RSA Components[12]

For RSA, there are two components of RSA.

- Choice of public key and private key
- The encryption and decryption algorithm

RSA has two keys, public key for encryption, and private key for decryption. In order to choose public key and private key, doing the following flow is necessary.

- Choose two large prime numbers, p and q
- Compute $n = pq$ and $z = (p-1)(q-1)$
- Choose a number, e, less than **n,** and **e** and z with no comMon factor
- Find a number, d, such that $ed - 1$ is exactly divisible by z

- The public key of RSA is ***(n,e)***, and the private key of RSA is ***(n,d)***

After get the keys the RSA flow shows as following.

- Ciphertext **=** ***C,*** and Plaintext = ***P.***
- Public key is ***(n,e)*** and Private key is ***(n,d).***
- Encryption ***:*** $C = P^e \bmod n$.
- Decryption ***:*** $P = C^d \bmod n$.

2.5.2 Complexity of RSA

Before explain the complexity of RSA, we define one term as a basic unit to estimate result.

- ***Define :***

  1-bit addition or subtraction as a unit computation

In RSA, n times of **modular-multiplications** are required. Suppose ***(n,e)*** both are n bits, the RSA compute at least n times **modular-multiplication**. And each **modular-multiplication** needs total n^2 times unit computation because one multiplication and one division are required. As a result, RSA needs $O(n^3)$ of computation.

There are two ways to speed up the RSA. The two algorithms are not conflict with each other. They can cooperate to speed up the RSA. They are **Fast Exponentiation Algorithm** and **Montgomery Algorithm**.

2.5.3 Fast Exponentiation Algorithm

Fast Exponentiation Algorithm is a trick to reduce the number of **modular-multiplication** from e times to log e. As previously described, we need at least n times **modular-multiplication**. However, it is not necessary to compute at least n times

modular-multiplication. For example, if we want to compute 8^{16} mod 13. We can first compute 8^2 mod 13 with one **modular-multiplication.** Then, 8^4mod 13 come from by one **modular-multiplication** of 8^2 mod 13 and so on. As a result, 8^{16}mod 13 needs only four times of **modular-multiplication**. We reduce the number of **modular-multiplication** form e to log e. This is the concept of **Fast Exponentiation Algorithm**. A real case lists as follows.

$11^{13} \bmod 53$

$11 \bmod 53 = 11$

$11^2 = 121, \quad 11^2 \bmod 53 = 121 - 2\times 53 = 15$

$11^4 = (11^2)^2$, $11^4 \bmod 53 = 15^2 \bmod 53 = 225 \bmod 53 = 225 - 4\times 53 = 13$

$11^8 = (11^4)^2$, $11^4 \bmod 53 = 13^2 \bmod 53 = 169 \bmod 53 = 169 - 3\times 53 = 10$

Therefore $11^{13} \bmod 53 = 11 \times 13 \times 10 = 1430 \bmod 53 =$

$1430 - 26\times 53 = 52$

The **Fast Exponentiation Algorithm** applied to RSA is shown in Fig.2-14. The gray box is a **modular-multiplication**. P is plaintext. The bit notation of e in public key ***(n,e)*** is $\{e_k, \ldots, e_2, e_1, e_0\}$.

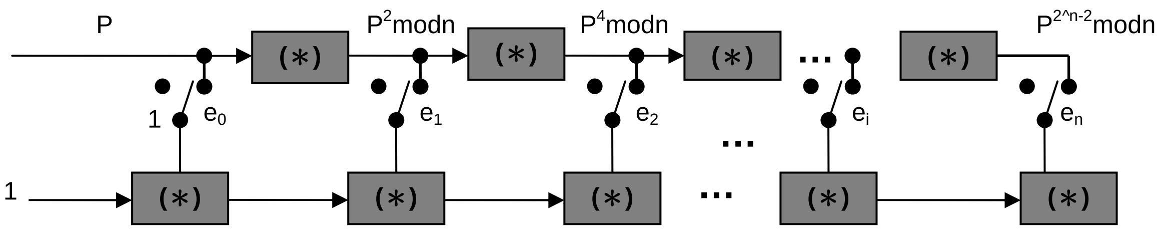


Fig. 2-14 Flow of RSA

The **Fast Exponentiation Algorithm** shows as follows.

```
Fast-Ex (P,E,N)
E, N < 2^r.
E = {e0, e1, e2, e3, e4,      …... en},    ei belong {1,0}.
{
        var t0 = 1;
```

```
        var t1 = P;
        for i=0 to n-1
        begin
            if( e_i = 1) then
                t0 = t0 × t1 mod N
            end if
            t1 = t1 × t1 mod N
        end for
        return t0;
    }
```

2.5.4 Montgomery Algorithm[13]

Fast Exponentiation Algorithm can reduce the **modular-multiplication** times in RSA. But computing a **modular-multiplication** still needs a lot of computation. In 1983, L.P. Montgomery proposed an algorithm that can speed up **modular-multiplication**.

For Fix N > 1, select a radix R coprime to N and the computation of modulo R are inexpensive to process. Let R^{-1} and N' be integers satisfying $0< R^{-1} < N$, 0 < N' < R and $R R^{-1} - NN' = 1$. For the given number, we can quickly compute the $T R^{-1}$ mod N from T if 0< T < RN by the following algorithm REDC.

```
REDC(T)

m = (T mod R) N' mod R

t = (T + mN)/R

if(t > N)then return t-N else return t
```

The result of REDC(T) will have the following characteristic. First, mN ≡ TN**N'** ≡ -T mod R, so t is an integer. Second, tR ≡ T mod N so $t \equiv T R^{-1}$ mod N. Third, 0< T+ mN < RN+RN, so 0 < t < 2N If we choose R = 2, the REDC will modify as follows.

```
REDC(T)      (Radix 2)

T = T_n……T_2T_1T_0   T_i∈{1,0}
```

$N' = N'_n......N'_2N'_1N'_0 \quad N'_i \in \{1,0\}$

$m = T_0 \times N'_0 \bmod 2$

$t = (T + mN)/R$

if(t > N)then return t-N else return t

The -N'N mod 2 is equal to 1 and N_0 is equal to 1 because N is the product of two large primes. As a result, N' mod 2 is equal to 1. This will lead ***m*** equal to T_0. We combine the multiplication and REDC(T) will get the **Montgomery Algorithm** shows as follow.

```
Mon ( A, B, N) (radix 2)
A , B , N < 2^r .
A = {A_r, ··· A_2 , A_1,A_0}    A_i ∍ {1,0}
B = {B_r , ··· B_2 ,B_1 B_0}    B_i ∍ {1,0}
{
    var ACC = 0;
    for i=0 to r do
        ACC = A_i×B+(A_iB_0^ACC_0)×N >>1
    end for
    return ACC;
}
```

The result of **Montgomery Algorithm** is $A\times B\times(2^n)^{-1}$mod N. And the **Montgomery Algorithm** needs only n^2 times of **unit computation.**

How does the **Montgomery Algorithm** reduce the computation? The Mon function has is a characteristic. If we process A to $A\times 2^n$ as ***A*** by doing Mon(A, 2^nmodN, N), the result of Mon(***A***,***A***,N) will be $A^2\times 2^n$ mod N, as $\boldsymbol{A^2}$. The preprocess can be done by Mon(A,2^{2r},N). With the same way, we can get $\boldsymbol{A^4}$, $\boldsymbol{A^8}$ …$\boldsymbol{A^{2\wedge i}}$, … $\boldsymbol{A^{2\wedge n}}$ which needs only one Mon. ($\boldsymbol{A^{2\wedge i}}$ is $A^{2\wedge i}\times 2^n$ mod N) To get the $A^{2\wedge i}$mod N we need the post processing, Mon($\boldsymbol{A^{2\wedge i}}$,1, N). We call the preprocessing as mapping, and the post processing as remapping. This characteristic makes the number **unit computations** in modular-multiplication from $2n^2$ to n^2.

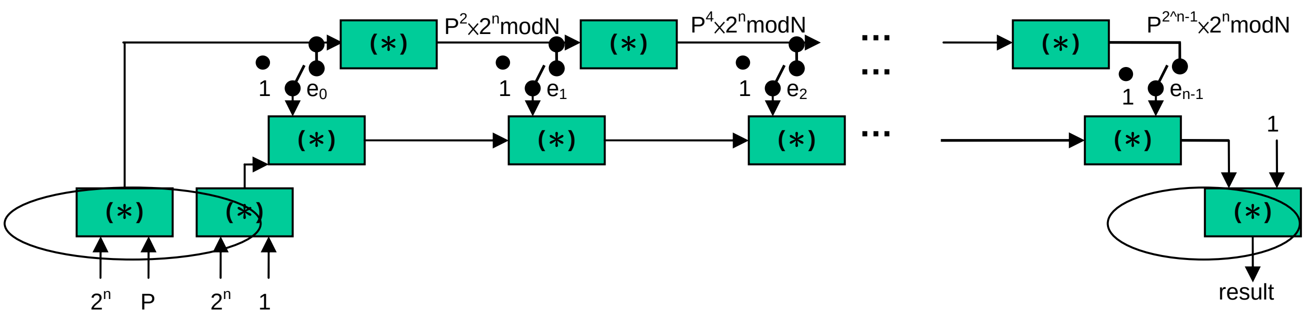


Fig. 2-15 Modified RSA flow

The flow applies **Fast Exponentiation Algorithm** and **Montgomery Algorithm** shows as Fig.2-15. The green box is a Montgomery Algorithm. The green box in circle in the left side is mapping. The right side is remapping. The algorithm of new RSA flow is following.

```
RSA(M, E, N)
E = { e_r,  … e_3, e_4, e_2 , e_1, e_0},    e_i ∍ {1,0}.
{
        t=2^{2r} mod N;
        P = Mon( t, M, N);          (Mapping)
        R = Mon( t ,1, N);          (Mapping)
        for i=0 to r-1 do
                if( e_i = 1) then
                        R = Mon( R, P, N);
                end if
                P = Mon( P, P, N);
        end for
        R = Mon(1, R, N);           (Re-mapping)
        return R;
}
```

2.6 Some Hardware Implementations

There are many researches about cryptographic hardware design. We classify them into two categories, Integrated Design and Dedicated Design. The Integrated design is able to perform more than two cryptographic algorithms. The dedicated design is dedicated to only one cryptographic algorithm.

2.6.1 Integrated Design

The Cryptonite processor is designed to provide a better tradeoff between flexibility and performance/area/power in the embedded systems, especially networking systems. It is a programmable architecture dedicated to cryptographic applications namely DES/3DES, AES, RC6, IDEA, MD4, MD5, SHA-1.

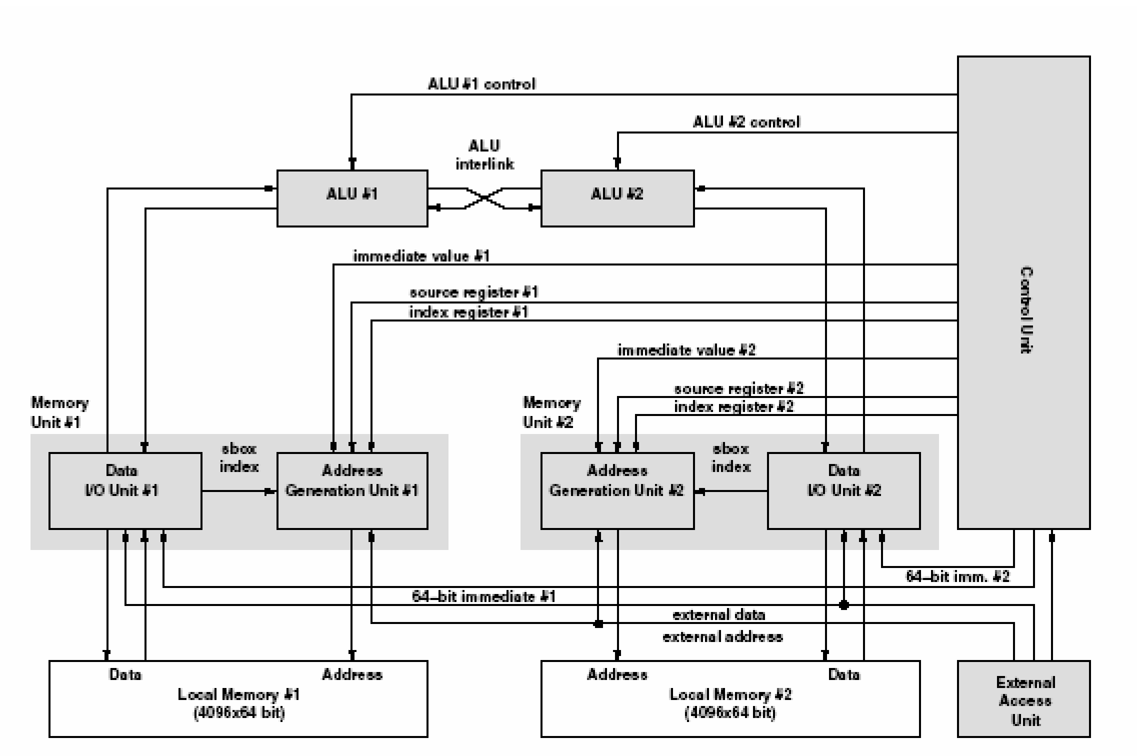


Fig. 2-16 Cryptonite Architecture

Architecture of Cryptonite shows as Fig. 2-16. It is two independent computing clusters, one for encryption and another for on-the-fly key generation. In general, key generation is independent to en/decryption flows. Coarse-grain parallelism can be exploited. On-the-fly key generation is vital for embedded systems solutions because storing/retrieving the round key needs for hundreds or thousands of cycle is not feasible. As a result, Cryptonite provides two independent computing clusters. There are four components, **Control Unit**, **ALU**, **Address Generation Unit** and **Data I/O Unit** in one computing cluster.

The Control Unit puts all other units on hold and grants the External Access Unit access to the internal data paths. Besides control other unit, Control Unit also provides 16 register for loop and control branch.

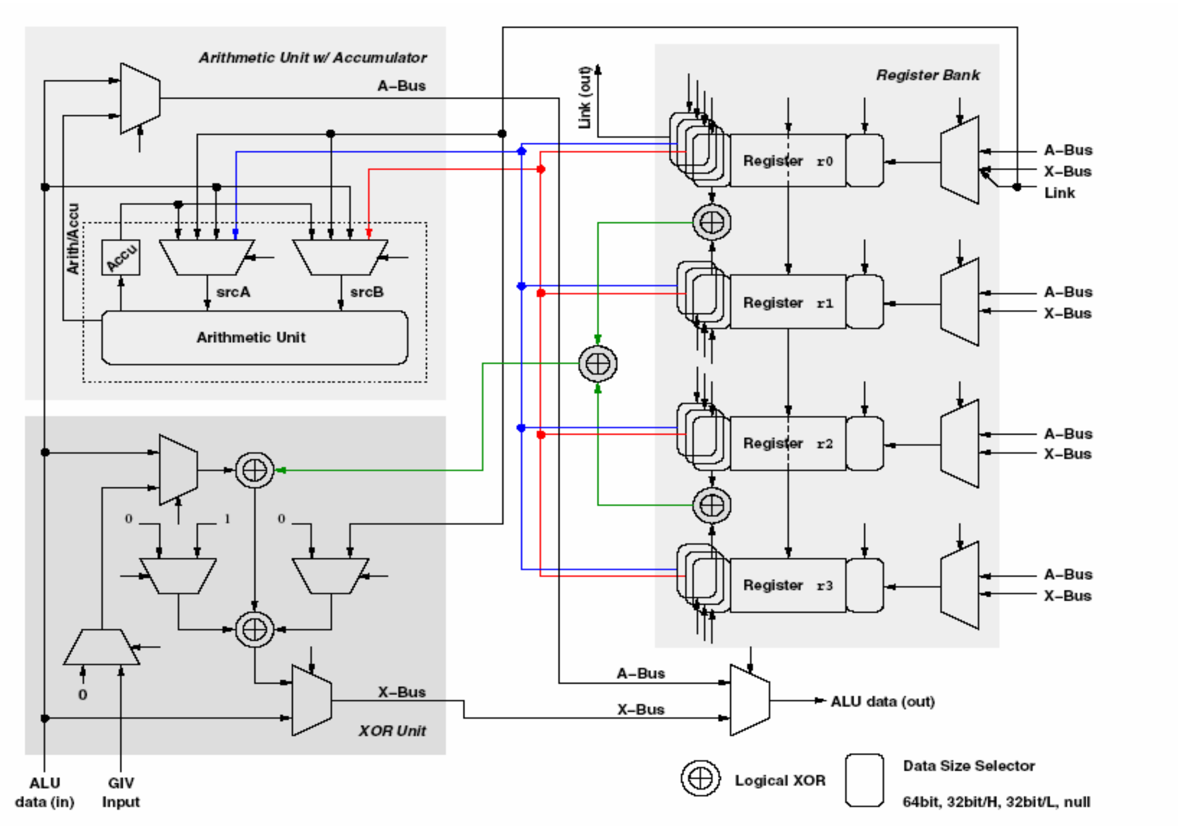


Fig. 2-17 ALU of Cryptonite

The ALU of Cryptonite shows as Fig. 2-17. There three main part of Cryptonite, register file, Arithmetic Unit and XOR unit. The register file is consisted of four 64-bit register. In order to compensate for the low register count, each register can be either used as one 64-bit or 2 32-bit registers. The AU supports conventional arithmetic operations, boolean operations and specialized functions supporting certain algorithms. The XU can provide six operand XOR.

The Address Generation Unit and Data I/O Unit are used to access local memory. The

Address Generation Unit contains small add/sub/and ALU for address generation. It supports eight addressing modes. The Data I/O Unit contains two data buffer for input/output. Data I/O Unit also contains a specialized DES unit. Fast DES execution not only needs highly specialized operations but also S-Box access to memory. Hence the DES is dedicated in the Data I/O Unit rather than ALU.

We compare our proposed design with Cryptonite in the following table.

		Cryptonite	Proposed design
Hardware organization		ASIP	Reconfigure Hardware
Parallelism		Coarse grain	Fine grain
AES	Key gen.	On the fly	Generate previously
	Implementation	Integrate to other algorithm	Integrate to other algorithm
DES	Key gen	On the fly	On the fly
	Implementation	Dedicated module	Dedicated module
Support		DES/3DES, AES, RC6, IDEA, MD4, MD5, SHA-1	DES/3DES, AES,RSA
Environments		High-end system	Low-end system
Flexibility		Cryptonite > Proposed design	
Area		Cryptonite > Proposed design	
Speed		Cryptonite < Proposed design	

Table 2-5 Comparison of Cryptonite and proposed design

Besides the cryptonite other industrial implementation will list in Appendix A-2

2.6.2 Dedicated Design

AES

[15] proposed a special purpose ASIC processor that implements the AES. In its implementation, there is only one hardware for en/decryption round and re-use the same piece to complete the whole en/decryption. Besides hardware reusing, the processor is also designed to complete one encryption round in one clock cycle. Furthermore, it uses on-the-fly key generation for encryption and generating the all key beforehand for

decryption. For SubBytes, it uses ROM to implement the SubBytes.

Its architecture is shown as Fig. 2-18. There are two main modules for AES. One for encrypt another for Key Expansion. Besides the two processing components, there are three interfaces. The Processor FSM is top-level controller interfacing with the user module. The input and output FSM is used to control input and output channel.

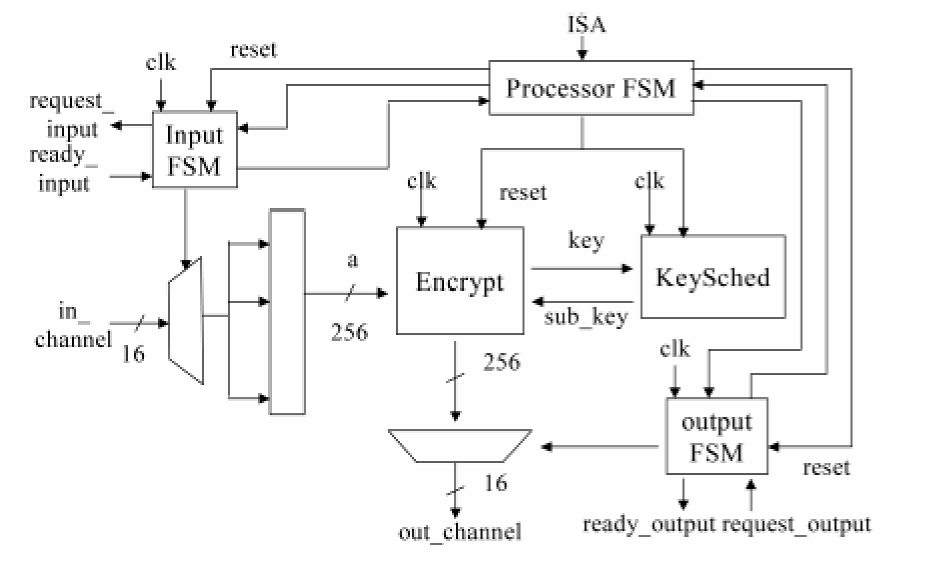


Fig. 2-18 Organization of [15]

Besides the research about the architecture of AES, there are some researches which focu on SubBytes. In [16], they propose a compact S-Box based on **composite field**. Using the composition field ,we can get smaller area of S-box.

	H.Kuo	**A. Satoh**
Unroll loop	No	No
SubByte	ROM	Dedicated Design
Area	H.Kuo > A. Satoh	
Speed	H.Kuo < A. Satoh	

Table 2-6 Comparison of AES ASIC

DES

In [17], this paper presents two FPGA implementations of DES. Both permit different pipeline levels with 21 and 37 cycles. As the section 2.4 shows, DES has some combinations and permutations operation. They are IP, IIP, E, and P in the Fig. 2-18. The S means S-box. The ⊕ means XOR. R means E^{-1} which is an inverse operation of E. The two new pipeline of DES are shows as follows. In the Fig 2-18, the left side is DES core one with 21 pipe stages, another is core two with 37 pipe stages. [17] wants to reduce the critical path and provide higher throughput by reorder the E in the DES flow.

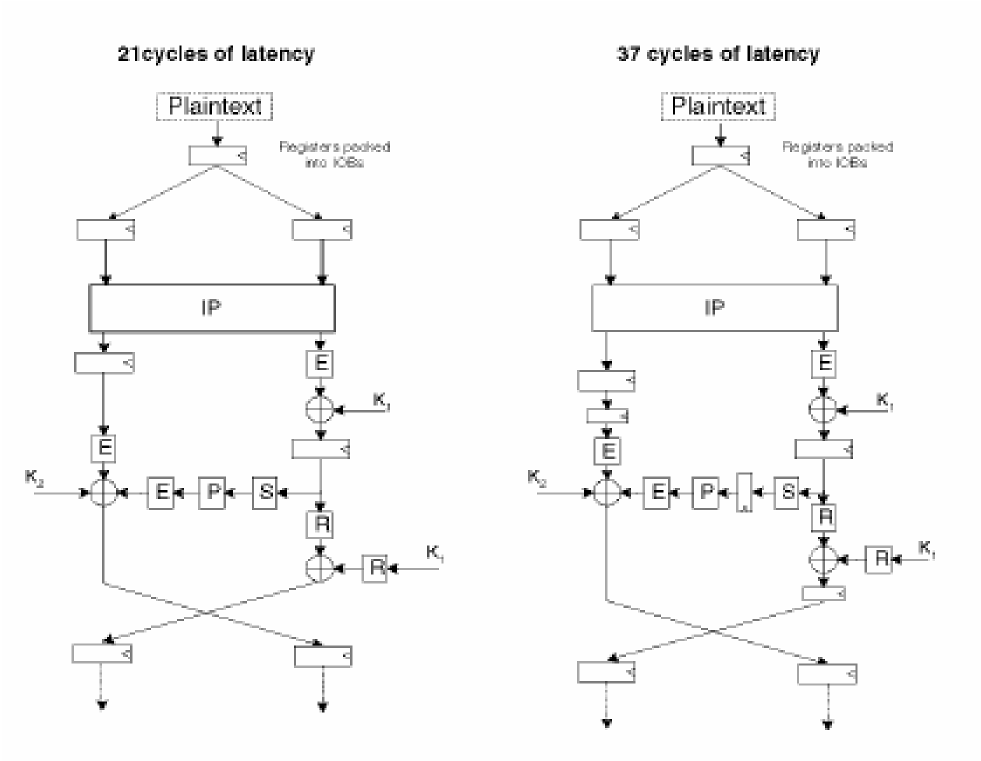


Fig. 2-19 The new DES pipeline

	Rouvroy -1	Rouvroy -2
Unroll loop	Yes	Yes
Pipe stages	37	21
Area	Rouvroy -1 > Rouvroy -2	
Speed	Rouvroy -1 > Rouvroy -2	

Table 2-7 Comparison of DES ASIC

RSA

The **Montgomery Algorithm** has a modified version. Unlike the original one, Mon(A,B,N) A,B,N < 2^n, which adds N along with partial product of A×B, the modified one adds N after finishing A×B. In **Modified Montgomery algorithm,** it splits **modular-multiplication** operation into multiplication procedure and Montgomery modular reduction procedure. Modular reduction procedure adds N 2n times to the accumulated result which is produced from the multiplication procedure.

In [18], they also based on the enhanced **Modified-Montgomery Algorithm**. Unlike previous implementation, they use one linear Bit-level Cellular-Array Design to perform the two procedures. The linear Bit-level Cellular-Array is Baugh-Wooley 2's complement array multiplier.

In [19], it based on original **Montgomery Algorithm**. Its architecture shows as follows. It is consisted of a linear Processing element (**PE**). Each **PE** performs fix length addition. Use these **PEs** to perform the all addition of RSA. The all architecture is like serial parallel multiplier.

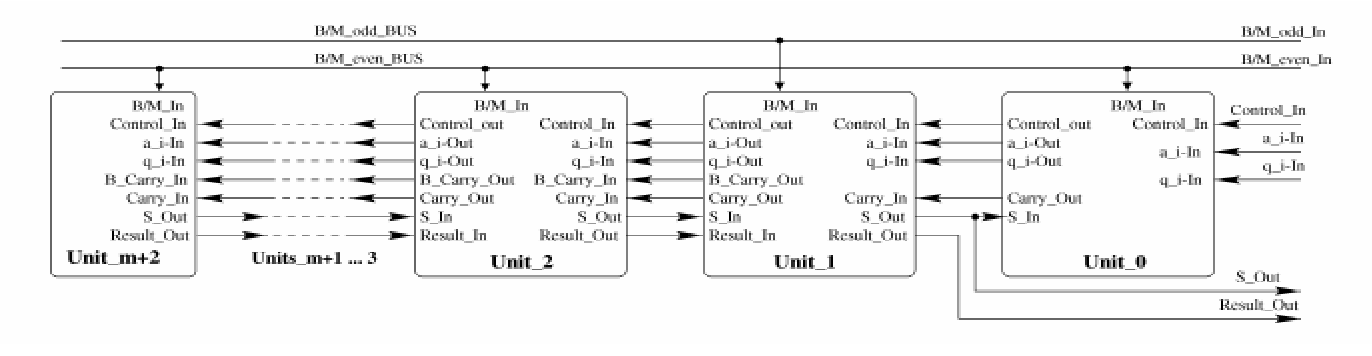


Fig. 2-20 Organization of [19]

	Blum & Paar	Su & Hwang
Mon	**Original**	**Modified**
Organization	**Serial-parallel multiplier**	**Baugh-Wooley multiplier**
Area	**Medium**	**Large**
Speed	**Medium**	**High**

Table 2-8 Comparison of RSA ASIC

Chapter 3 Design

In this chapter, we first summarize the all operation of the three cryptographic algorithms by its characteristic and classify them into three categories. Then design the components for the categories respectively.

3.1 Function Requirement

Cipher	Function		Comment
AES	**Key expansion**	**XOR**	**8 bits xor**
		Table look up	**8 bits in/8 bits output.**
		Rotate Byte	**Rotate left 1 bytes**
	Round	**X_TIME**	**Special operation for Galois field**
		XOR	**8 bits xor**
		Table look up	**8 bits in/8 bits output.**
		Rotate Byte	**Rotate right or left 1, 2, and 3 bytes**
DES	**Key scheduling**	**PC-1**	**Permutation bits from 64 bits to 56 bits**
		PC-2	**Permutation bits from 56 bits to 48 bits**
		Shift 1 or 2	**Shift left and right 28 bits one or two bits individually**
	Round	**Expand *E***	**Expand input form 32 bits to 48 bits**
		Permutation *P*	**Permutation 32 bits**
		XOR	**48 and 32 bits XOR**
		Table look up	**6 bits in 4 bits out**
RSA	**MON**	**Addition**	
		Shift	**Shift right one bits**

Table 3-1 Function requirements

At Chapter 2, we introduce for the three cryptographic algorithms. At this Chapter we first summarize the function requirement of the three algorithms in Table 3-1. In table 3-1, the function requirements for the three cryptographic algorithms are listed. These function requirements can be classified into three kinds depending on their characteristic. In Table 3-2, shows the three categories.

Class	*Type*	*Algorithm*	*Comment*
Permutation & Combination	**Expand**	DES	32 bits to 48 bits
	Permutation function	DES	32 bits permutation
	Initial permutation	DES	Start of the flow 64 bits to 64 bits
	Inverse permutation	DES	End of the flow 64 bits to 64 bits
	Rotate Bit	DES	Rotate left one or two bits
	Permutation Choice 1	DES	64 bits to 56 bits
	Permutation Choice 2	DES	56 bits to 48 bits
	Rotate Byte	AES	byte level rotate left or right
	Shift	MON	Shift one bit
Computation	**XOR**	DES,AES	8 or 32 bits xor
	Xtime	AES	special operation for Galois field
	Addition	MON	
Memory	**Buffer**	DES,AES,MON	Memory output buffer
	Look-up table	DES,AES	Table look up

Table 3-2 Operation classification

These three categories are **Permutation & Combination**, **Computation** and **Memory**. **Permutation & Combination** are some fix wiring operation of the three cryptographic algorithms. **Computation** is the operators which are belong to arithmetic and logic. The three cryptographic algorithms need only Addition, XOR and X_TIME. The finial class is **Memory** which is table look up and data buffer in the three cryptographic algorithms. Because of the three classes, our system will have the three components to perform the three kinds of operation respectively.

Before design module for the three class operation, we notice that DES has some characteristics. First, DES is suit for on-the-fly key generation. So, the hardware cost for

DES key generation is small. Second, the hardware circuit for table look up in DES is small too. Because these two reasons, we dedicated part of DES for get better result.

3.2 System Overview

As the section 3.1 described, our system overview is shown in Fig. 3-1. We have six components. They are **Permutation & Combination Unit**, **Computation Unit**, **Memory Unit**, **DES Unit, Context Memory/Decoder** and **Ctrl**.

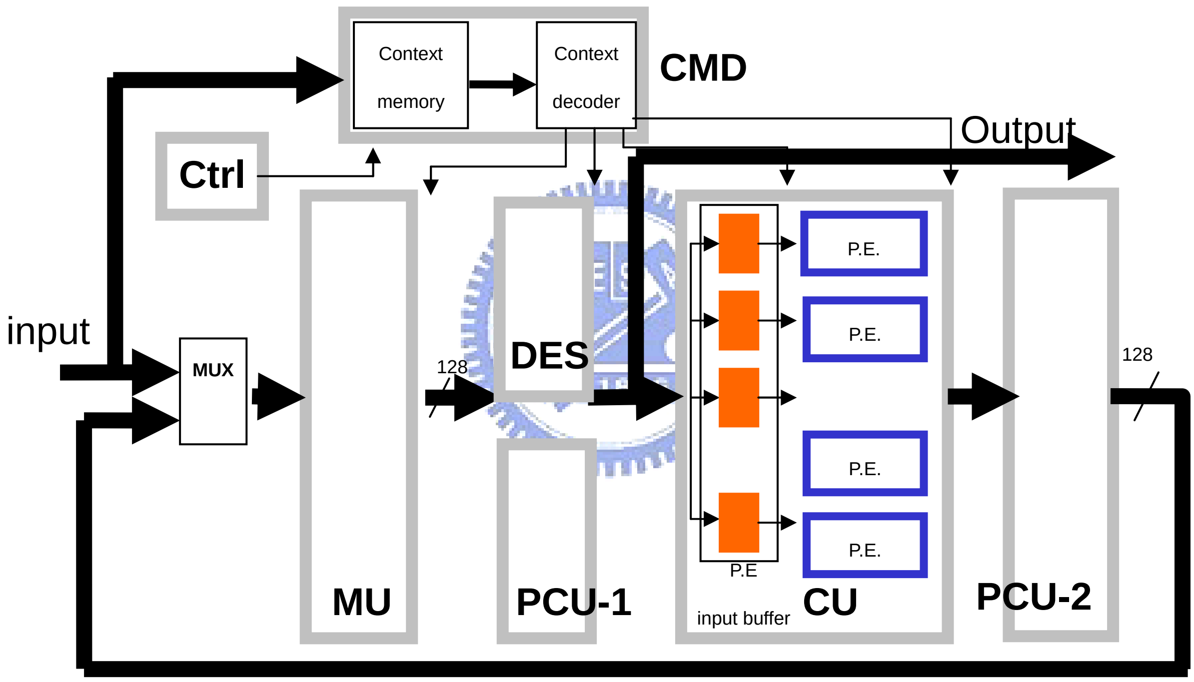


Fig. 3-1 System overview

Permutation & Combination Unit (PCU-1 or PCU-2) :

The **Permutation & Combination Unit (PCU)** executes the fix wiring tasks. As cryptographic algorithms go, PCU routes the data to **Computation Unit** or **Memory Unit**. In DES, there are many wiring operation, such like PC1, PC2, Expand and etc. The most operation wiring of DES is in this unit.

Computation Unit (CU) :

The **Computation Unit** performs arithmetic or logic operators of three cryptographic algorithms. It is composed by **processing elements (PE)** and some additional circuits for MON**.** Each **PE** is a functional unit which has its own register file which stores immediate data during execution. The additional circuits will introduce at section 3.4.4.

DES UNIT

This unit performs some DES operation to get better space-time product. It supports on-the-fly key generation.

Memory Unit (MU) :

As the **Montgomery Algorithm** goes, it needs to store large immediate data and preload data for smooth execution. Besides the **Montgomery Algorithm**, DES and AES also need to look up table. As a result, **Memory Unit** is used to store immediate data and preload buffer of MON and look-up table for DES and AES

Context Memory/Decoder (CMD) :

The three cryptographic algorithms needs different context to control **CU**, **PCU**, and **MU.** We use the **Context Memory** (**CM**) to store the context for each cryptographic algorithm. **Context Decoder** (**CD**) decodes the context to control signals.

Control Unit (Ctrl) :

Ctrl is the central control unit to control the whole hardware. It consists of a finite state machine, some flow registers, address register and address generation function. It uses the finite state machine to control our hardware and the flow registers to control the flow of the three cryptographic algorithms.

3.3 Permutation and Combination Unit

Permutation and Combination Unit is used to route data between **CU** and **MU**. In the following table, it shows the permutations and combinations perform by this unit.

If we use high flexibility implementation, such as GRP [20],OMFLIP[21], CROSS [22], and BFLY [23][24]. These will lead to high area cost. As a result, we just make our **PCU** to perform the fix permutation or combination wiring. It operation is shown in the Table 3-3

Unit	***Type***	***Algorithm***	***Comment***
PCU-1	**Initial permutation**	DES	Start of the flow 64 bits to 64 bits
	Permutation Choice 1	DES	64 bits to 56 bits
	Rotate Byte	AES	byte level rotate left or right
PCU-2	**Inverse permutation**	DES	End of the flow 64 bits to 64 bits

Table 3-3 Permutation and combination performed by PCU

3.4 Computation Unit

The **CU** performs tasks of logic operators of three cryptographic algorithms. The **CU** is composed by **processing elements (PE).** Each **PE** has its own register file to store immediate data. For **Montgomery Algorithm** consideration, **CU** needs additional circuits to execute **Montgomery Algorithm**. First we will introduce the **PE** design and then introduce these additional circuits in section 3.4.4.

3.4.1 Observation

Before design our **PE**, we first introduce the tasks of **PE** and some observations. For full parallelism, The hardware resource needed by AES and MON is shown in Table 3-4.

	AES_D (128)	**AES_E(128)**	**Mon(1024)**
XOR(8 bits)	**9×16×10+32**	**9×16×5+32**	**≥1024×128×2**
X_TIME(8 bits)	**9×16×6**	**9×16×1**	**0**

Table 3-4 Hardware requirements of the three cryptographic algorithms

As the section 2.3.5 and 2.3.6 described, the data flow graph of AES which combines the MixColumn and the AddRoundKey is shown in Fig.3-2. The AES-128 composed by 9 rounds and two additional AddRoundKey. As a result, the XOR requirements of en/decryption are **9×16×5+32** and **9×16×10+32** and X_TIME are **9×16×6** and **9×16×1**separately.

An addition is composed by two XOR. In Mon (1024), we know that it works like a multiplier. The original **Montgomery Algorithm** needs total 1024 times of 1024-bit addition. Now, we formalize 1024-bit addition to 128 times 8 bits addition. This is why the Mon needs at least 1024*128*2 time 8 bits addition.

As a result, the MON needs only Double XOR. It shows as in Fig. 3-3.

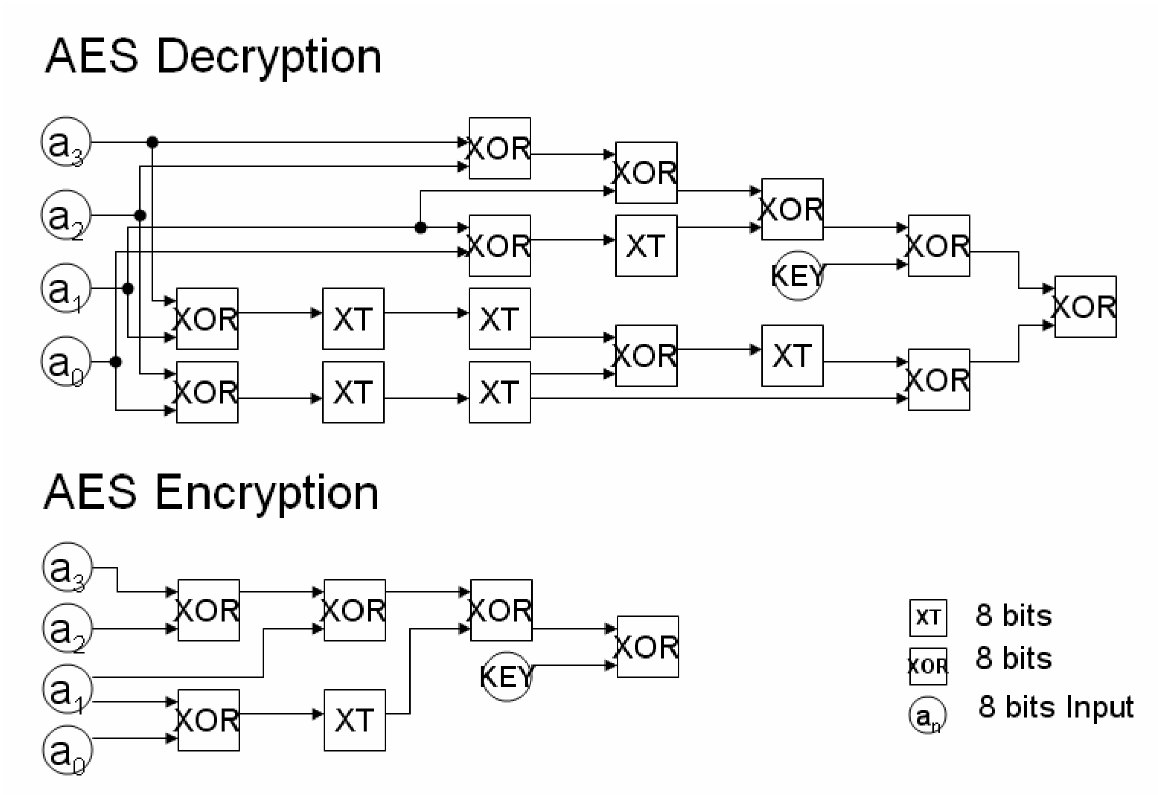


Fig. 3-2 En/Decryption data flow graph of Mixcolume and AddroundKey

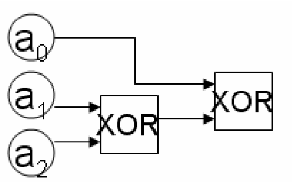


Fig. 3-3 MON data flow graph

So the computation granularities between the three cryptographic algorithms are shown in Fig. 3-4. AES has coarser granularity. MON have finer granularity.

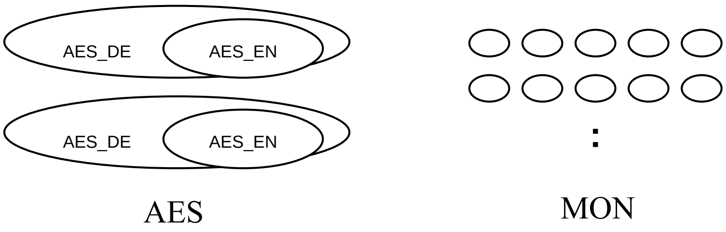


Fig. 3-4 Granularity of the two cryptographic algorithms

3.4.2 Organization of Computation Unit

As the observation of 3.4.1 described, the MON needs only simple hardware for execution. So, the Mon will interleave store and addition. As a result, too much complicated hardware will be useless. Our **CU** organization is shown in Fig. 3-5. It is a one dimension array of **PE**. The direction of data inputs stream is vertical to the one dimension array. Each **PE** has the ability to perform the three algorithms. The more **PE** have, the higher parallelism we have.

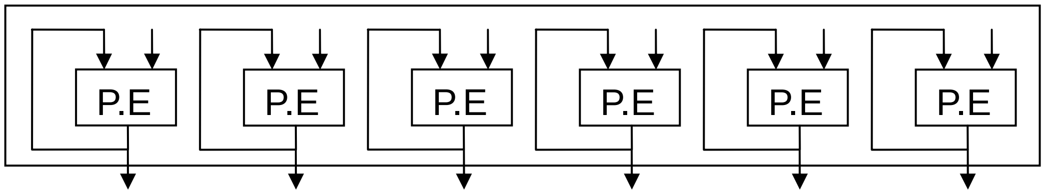


Fig. 3-5 Organization of CU

If we want to design an organization of **PE** which gets better **space-time product**, we must make hardware idle as less as possible. There are two kinds of design method to analysis data flow graph of the three algorithms.

- Partition AES to fit MON.
- Put MON into AES.

In the first method, it is finding a smallest hardware which can be divided by AES en/decryption, MON, like LCM. The second method is like to find GCD. Because the first method will produce smaller hardware unit, it will suit for embedded system to scale than hardware produced by second method. As a result, we choose the first methodology.

3.4.3 Design of P.E.

We have chosen the first method to analysis in order to get small **space-time product**. It seems no trivial solution. So, we have proposed a structural methodology to decide our **PE** organization. The concept our methodology is that

- Candidate Choosing : We choose some possible candidates of P.E. organization, then design it dedicatedly
- Evaluation : Evaluate the possible candidates

Candidate Choosing

Before introducing our methodology, we must define some terms previously. The organization of **PE** comes from OPSET.

Definition

- ***OP : A connected direct graph composed by XORS and/or X_TIMEs***
- ***OP_LEN : Number of nodes in critical path of OP.***
- ***OPSET i : A set of OPs that can consist the three data flow graph***

The principle of our OPSET ***i*** finding is shown as following.

1. ***First OP***
   - ***OP_LEN is i***
   - ***Most frequent OP and occur more than twice***
2. ***Other OPs***
   - ***Dependent on occurrence frequency***

In the following, XT means X_TIME, th four candidates of OPSETs lists as follows.

OPSET 1 (I)

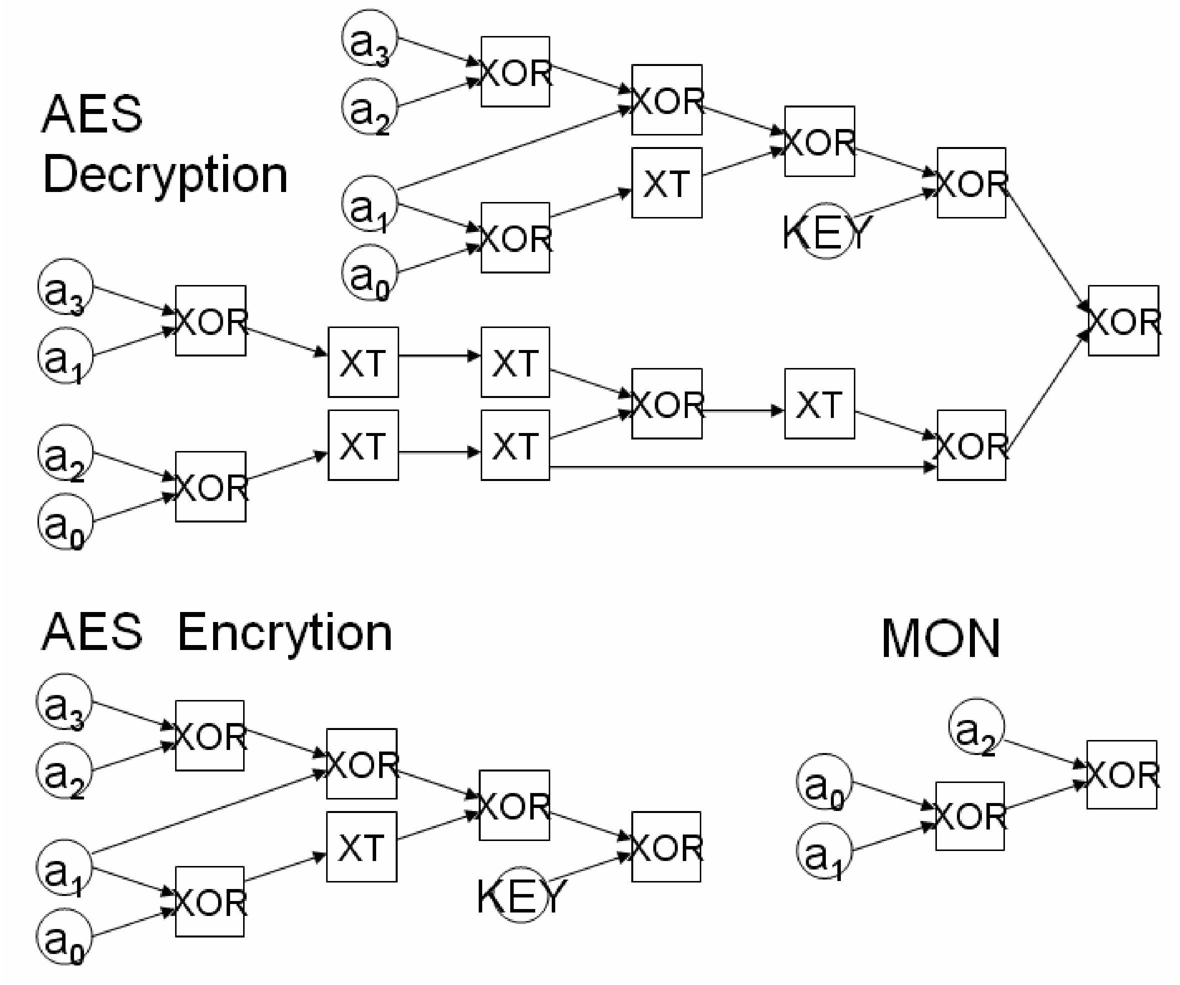


Fig. 3-6 Partition AES and MON

The OP set is

Fig. 3-7 OP set for OPSET 1

The P.E. organization of OPSET 1 is in Fig. 3-8.

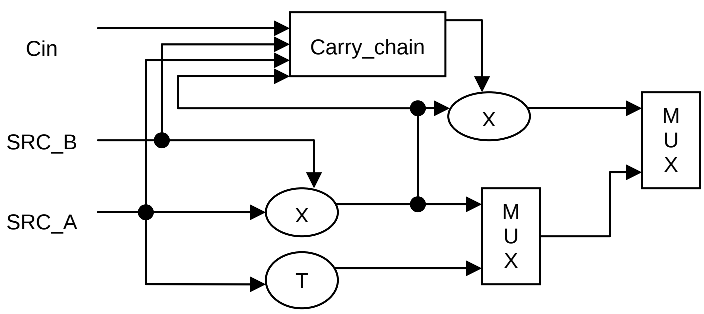


Fig. 3-8 OPSET 1 organization

OPSET 2 (II)

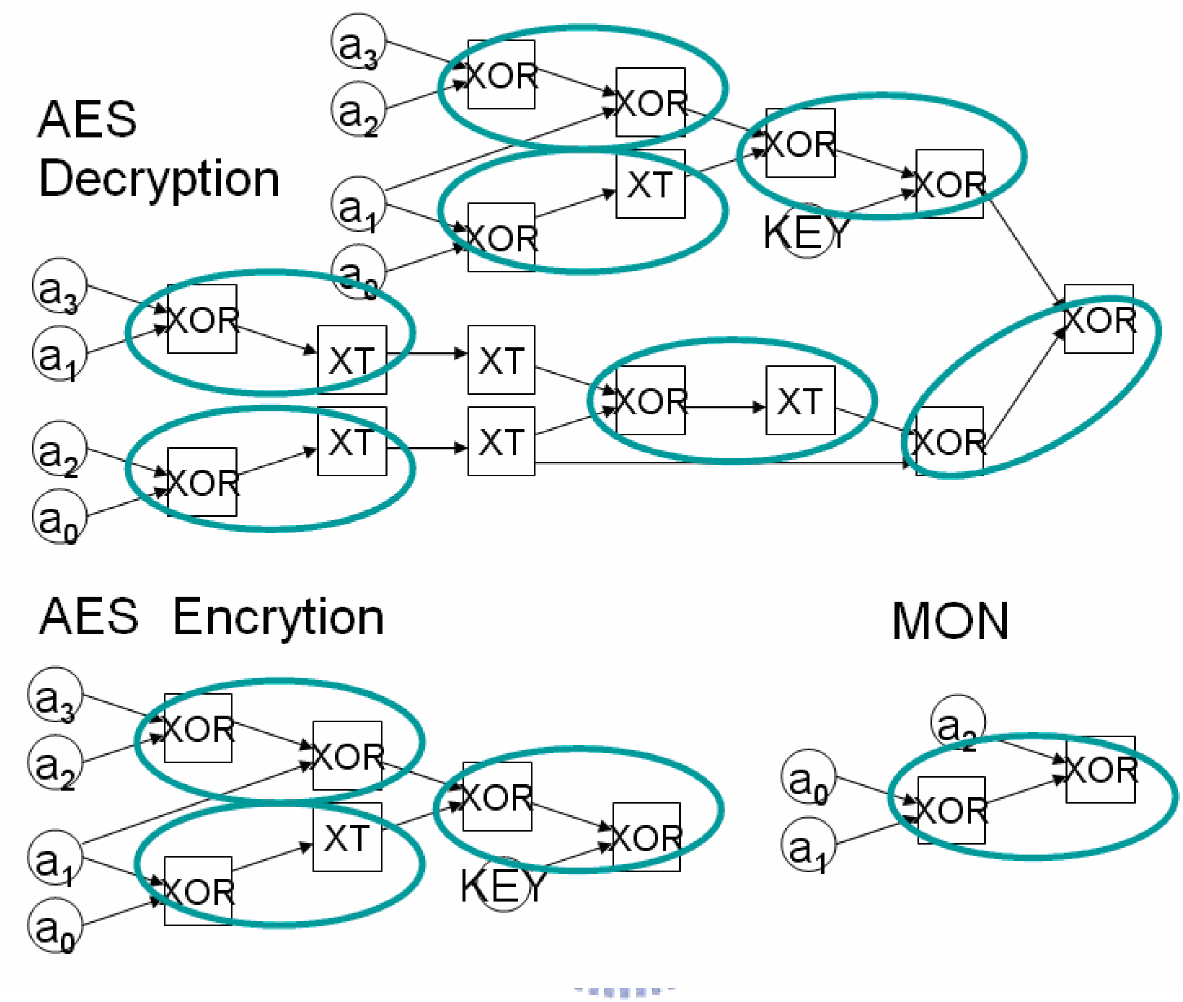


Fig. 3-9 Partition AES and MON

The OP set is

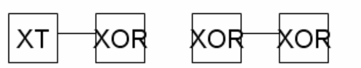


Fig. 3-10 OP set for OPSE 2

The P.E. organization of OPSET 2 is in Fig. 3-11.

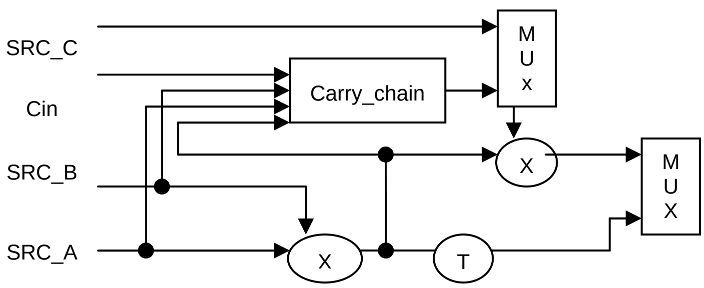


Fig. 3-11 OPSET 2 organization

OPSET 3 (III)

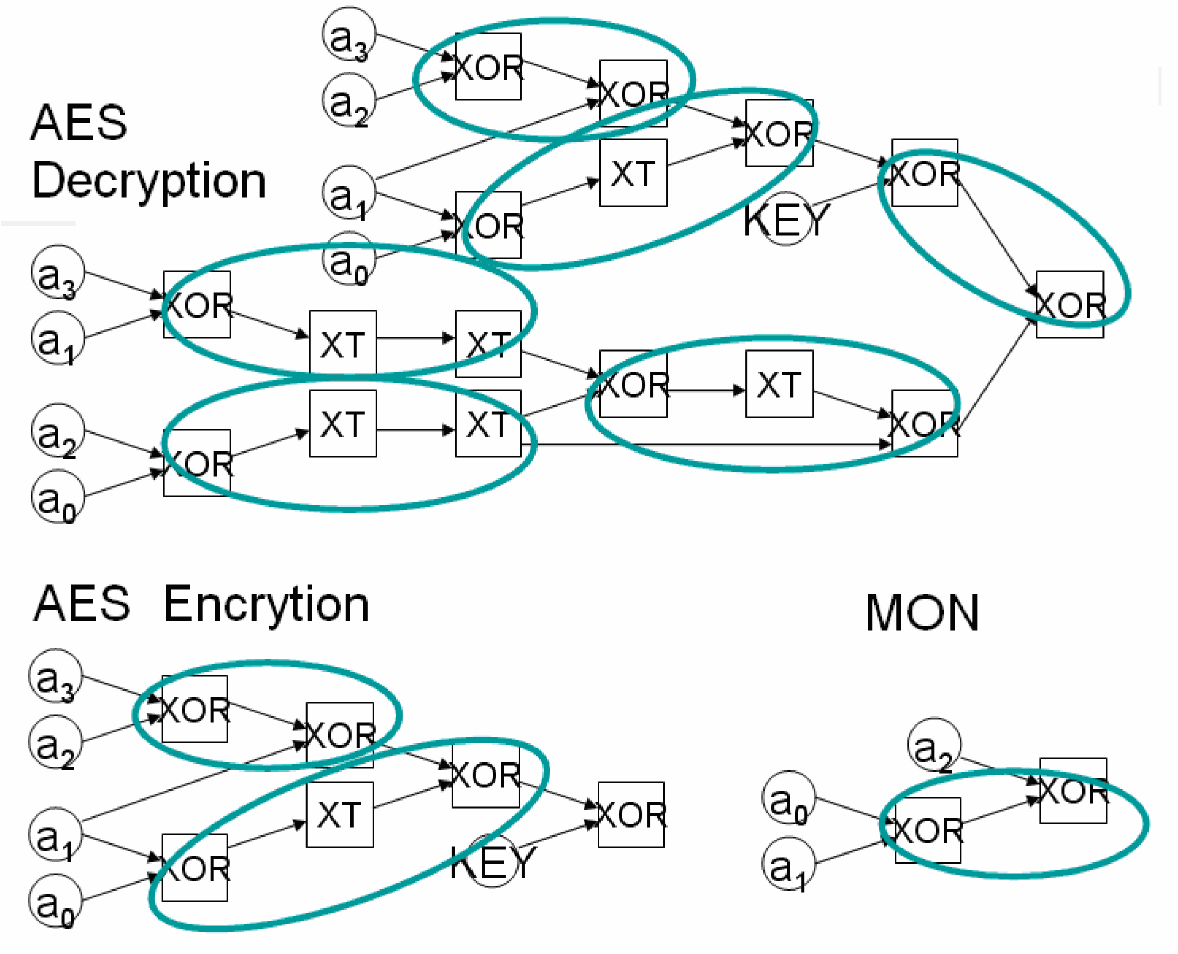


Fig. 3-12 Partition AES and MON

The OP set is

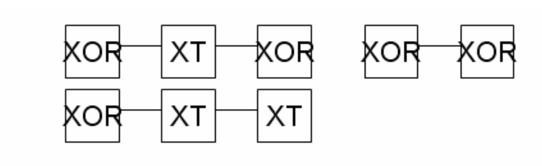


Fig. 3-13 OP set for OPSET 3

The P.E. organization of OPSET 3 is in Fig. 3-14.

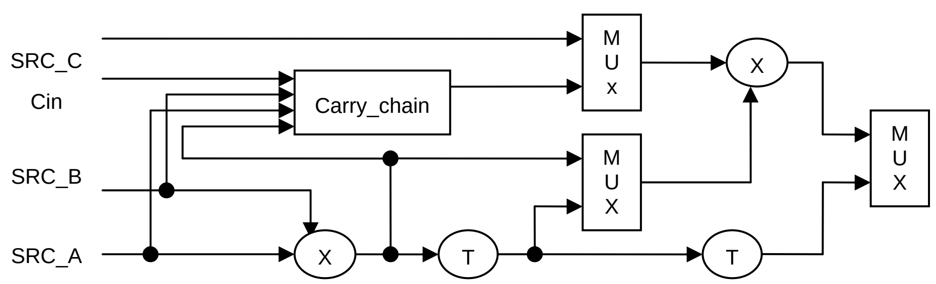


Fig. 3-14 OPSET 3 organization

OPSET 4 (IV)

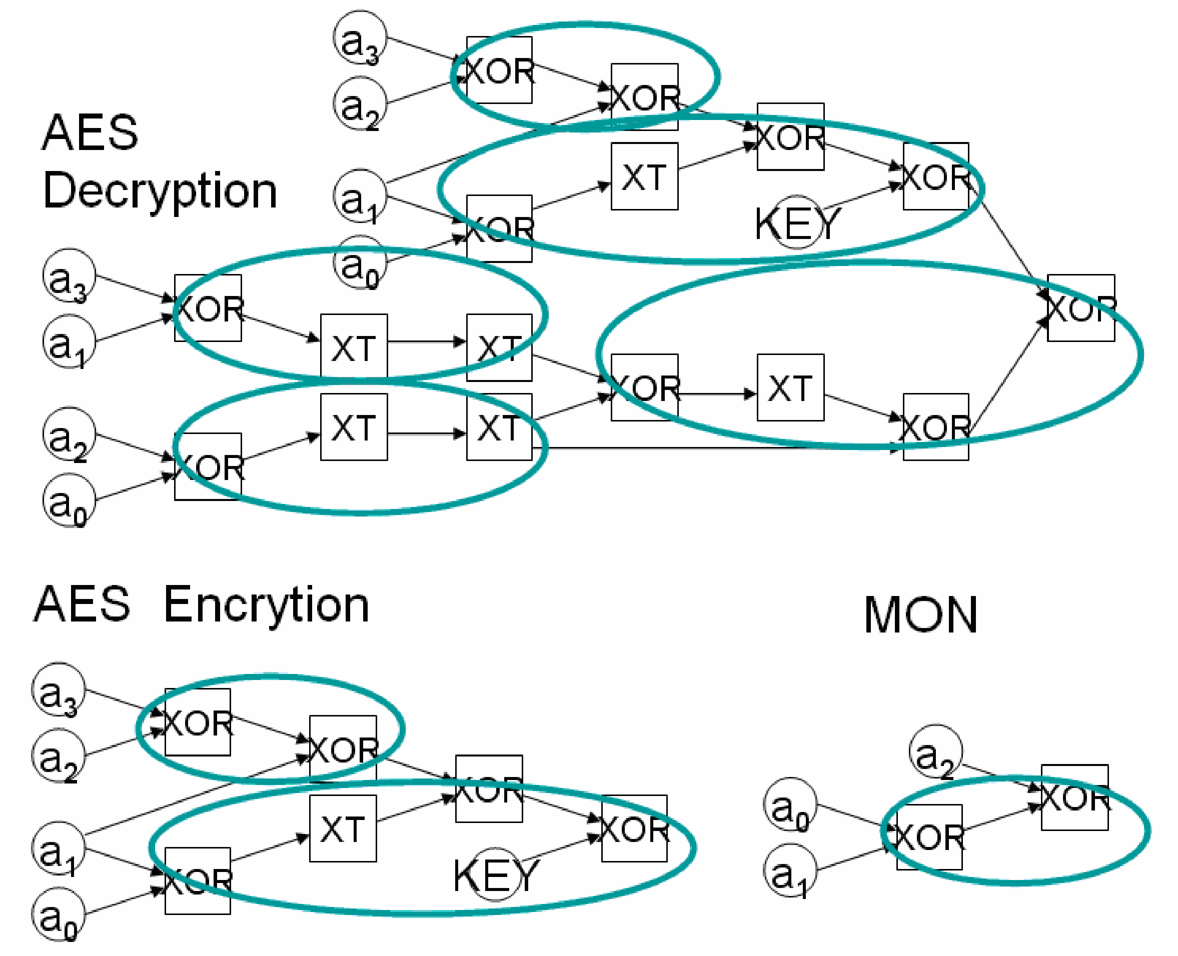


Fig. 3-15 Partition AES and MON

The OP set is

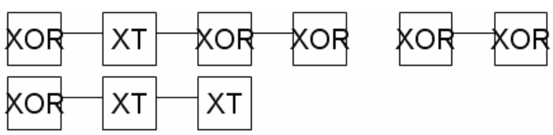


Fig. 3-16 OP set for OPSET 4

The P.E. organization of OPSET 3 is in Fig. 3-17.

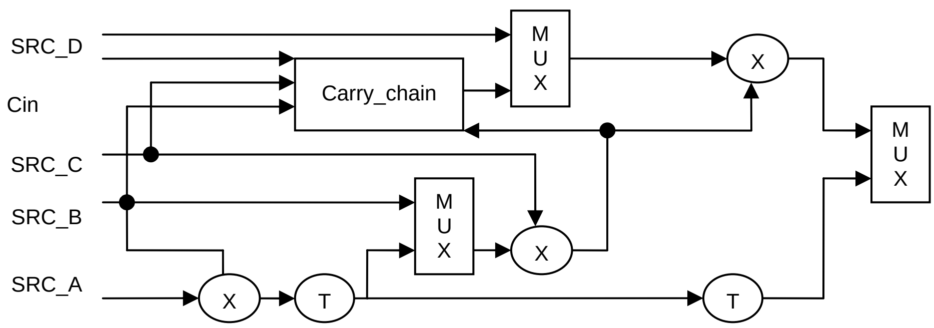


Fig. 3-17 OPSET 4 organization

Evaluation

According to previously choosing, we get the critical path and area of the four **PE** by synthesis these design. These information shows in the following table.

	I	II	III	IV
Area	1078.579	1228.222	1697.540	1910.574
Time	3.89	4.03	4.04	4.98

Table 3-5 Area and timing of the four OPSET

Use the four candidates **PE** to execute encryption and decryption of AES-128, AES-192, AES-256, MON-256, MON-512, MON-1024, MON-2048, and MON-4096. The execution cycles of these tasks are list in following table

	1	2	3	4
AES_128_D	2560	1440	960	800
AES_128_E	960	480	480	320
AES_192_D	3072	1728	1152	960
AES_192_E	1152	576	576	384
AES_256_D	3584	2016	1344	1120
AES_256_E	1344	672	672	448
MON_256	8192	8192	8192	8192
MON_512	32768	32768	32768	32768
MON_1024	131072	131072	131072	131072
MON_2048	524288	524288	524288	524288
MON_4096	2097152	2097152	2097152	2097152

Table 3-6 Cycles needed by different specification of cryptographic algorithms

With the area, timing and execution cycles, we can get **space-time product** of the four OPSET. In the following result, we don't consider the latency for accessing external memory.

In Fig. 3-18, it summarizes symmetric cryptographic algorithms **space-time product** of the four OPSETs.

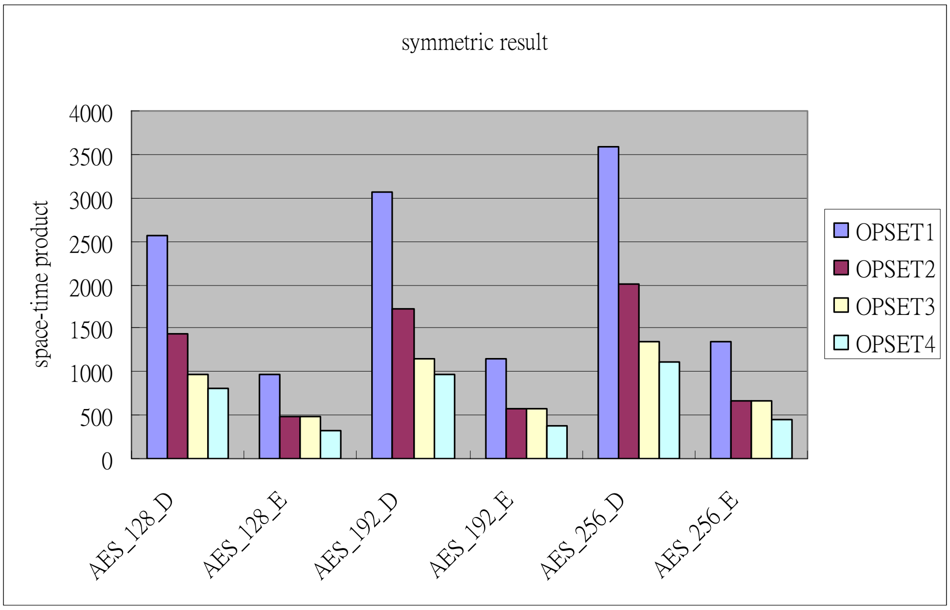


Fig. 3-18 AES space-time product of the four OPSETs

In Fig 3-19, it summarizes asymmetric cryptographic algorithms **space-time product** result of the four OPSETs.

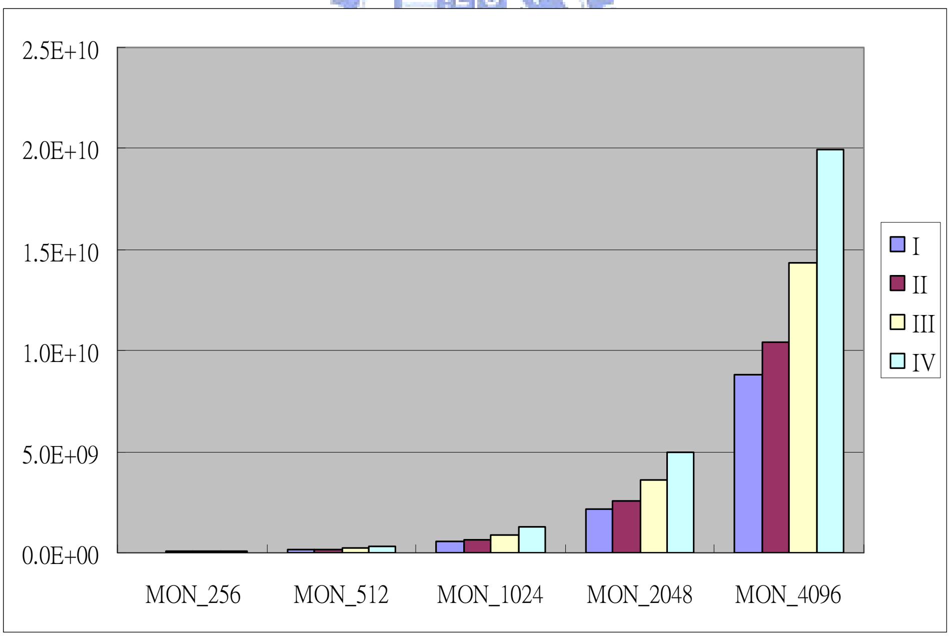


Fig. 3-19 MON space-time result of the four candidates

According to symmetric and asymmetric cryptographic algorithm usage in the cryptosystems, we use the following function to evaluate result. S_i means the symmetric result. A_j means the asymmetric result

$$\sqrt{\sqrt[n]{S_0 \times S_1 \times \ldots . S_n} \times \sqrt[m]{A_0 \times A_1 \times \ldots\ldots A_m}} \quad (1)$$

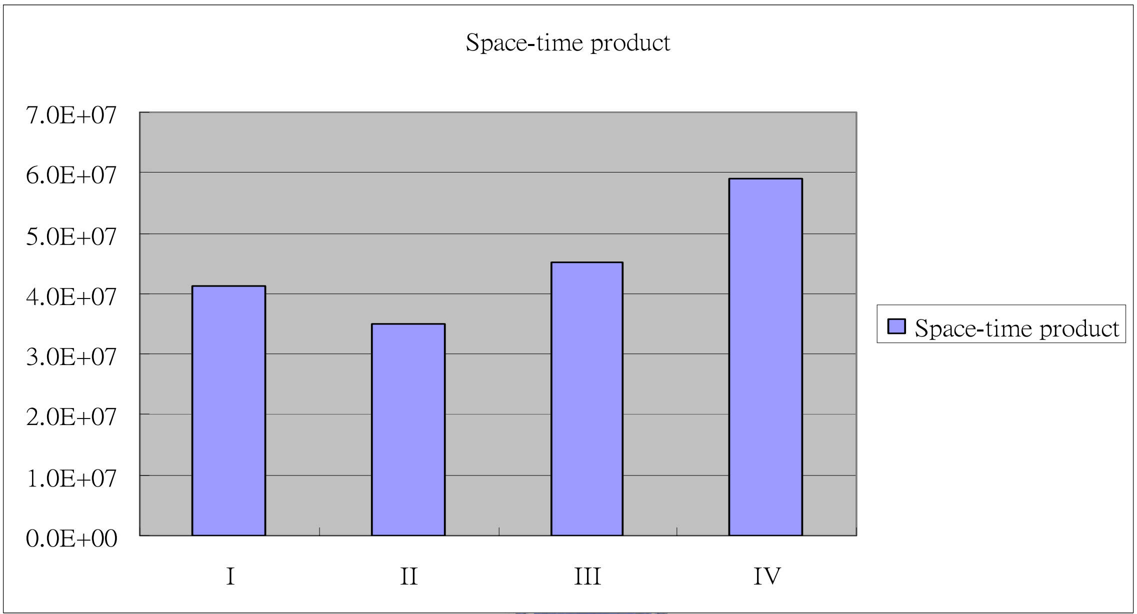


Fig. 3-20 Space-time result of the four OPSET

As the Fig. 3-20 showing, OPSET 2 will get best **space-time product** in the four OPSET**.** As a result, we will choose the organization of OPSET 2 as our **PE**. Besides the organization, we still need register file for **PE** to store a small amount of immediate result. OPSET 2 needs 6 registers to store. The total P.E. organization shows as following.

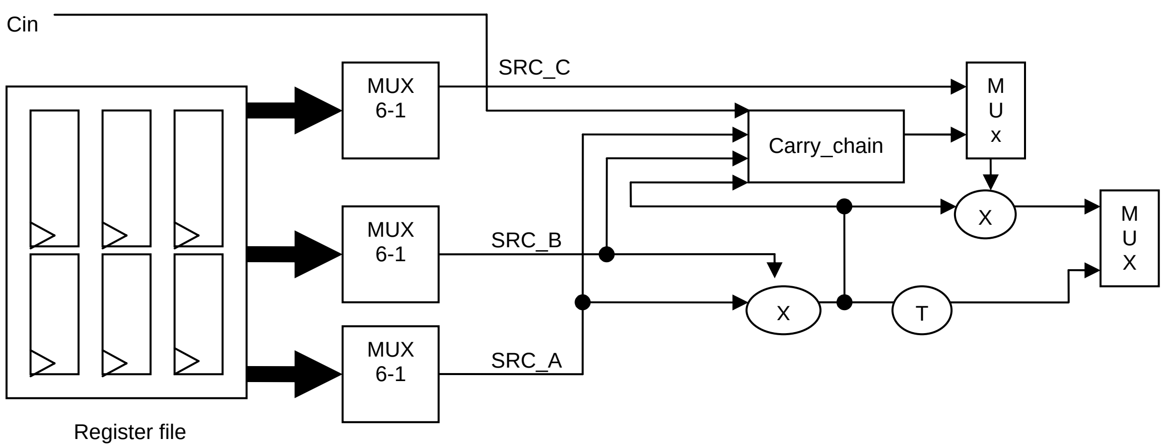


Fig. 3-21 Organization of PE

3.4.4 Additional hardware for MON

Our **PE** organization is designed in previous section. Our **PE** is able to perform AES. But for **Montgomery Algorithm**, there are some additional circuit requirements.

First, we need a carry chain to chain the long addition and shift chain to perform one bit shift during MON.

Second, we must depend on the least significant bit of result and the multiplier to decide which operand should be selected to add with accumulation result. So, we design the hardware, named Operand Select, to do this job.

Last but not least, overflow will occur as the MON goes,. If overflow occurs, the result of MON will be wrong. In order to prevent the condition, we design a one bit half adder to solve the problem. These additional circuit for Mon is shown as Fig. 3-22.

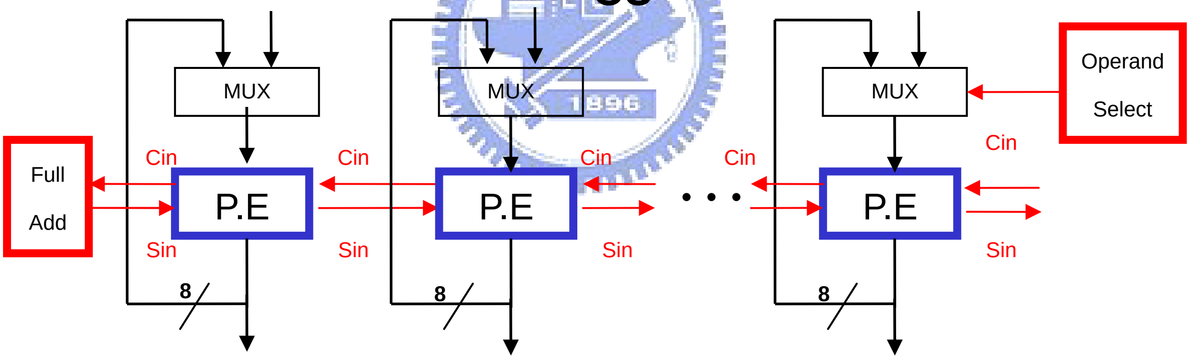


Fig. 3-22 Additional hardware

3.5 DES Unit

The organization of DES unit is shown as follows. It is able to perform a round of DES in two cycles and support on-the-fly key generation. It has two inputs one for en/decryption flow, named round i, the other key generation, named key gen. . Round i has two XOR, two permutation, expand and permutation and eight distinct look up table. For Key Gen., there are only two permutation, Left circuit Shift and PC2.

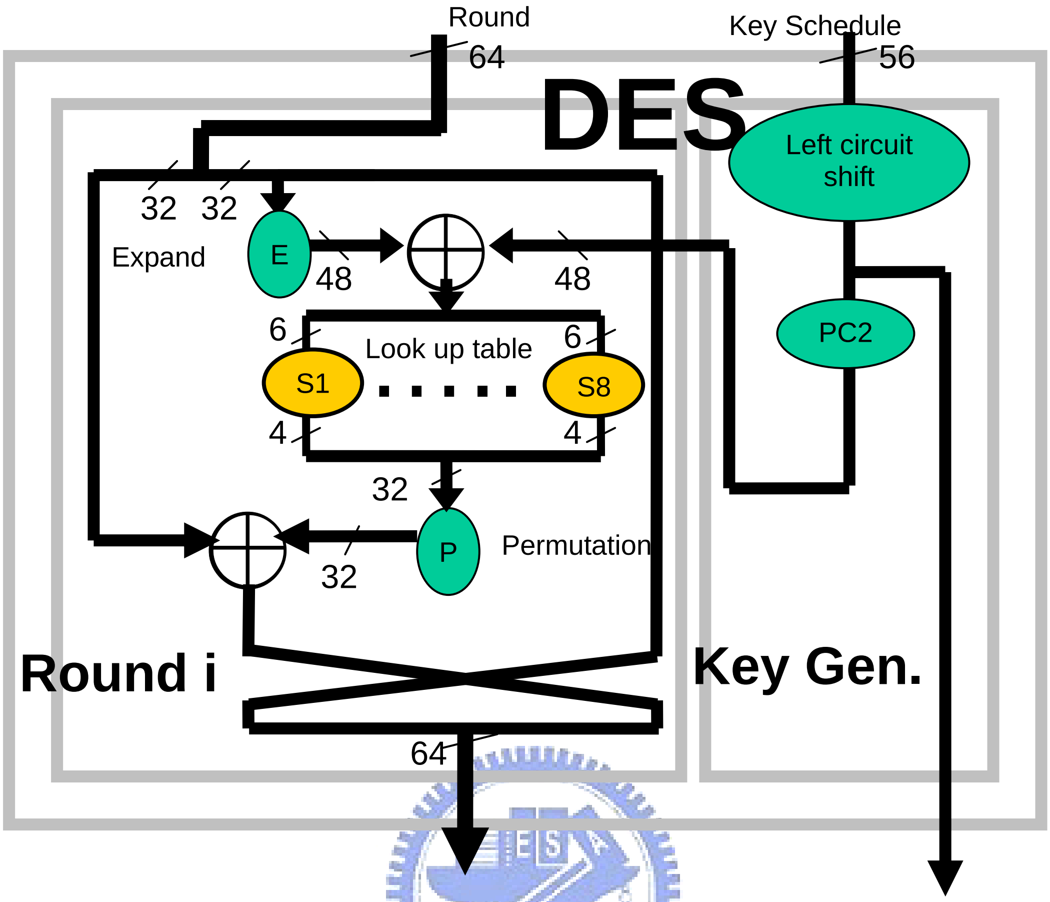


Fig. 3-23 Organization of DES unit

3.6 Memory Unit

In the previous section, we have finished the design of **PE** organization. During evaluate of the four OPSET, we don't consider the memory access latency. In this section, we will discuss it and design the input data buffer and analysis bandwidth requirement for the three cryptographic algorithms.

3.6.1 Organization

In order to support different security level for MON, variable length addition is unavoidable. To perform variable length addition, we partition addition into some fix length sections and then our system performs the fix length addition. As a result, MON needs to restore much data during execution. Besides data buffer for large storage, MON also needs some preload buffer to hide memory access.

Besides MON, we put look-up table of AES into the **MU**. In AES, it needs only one table which is 8-bit input 8-bit output table.

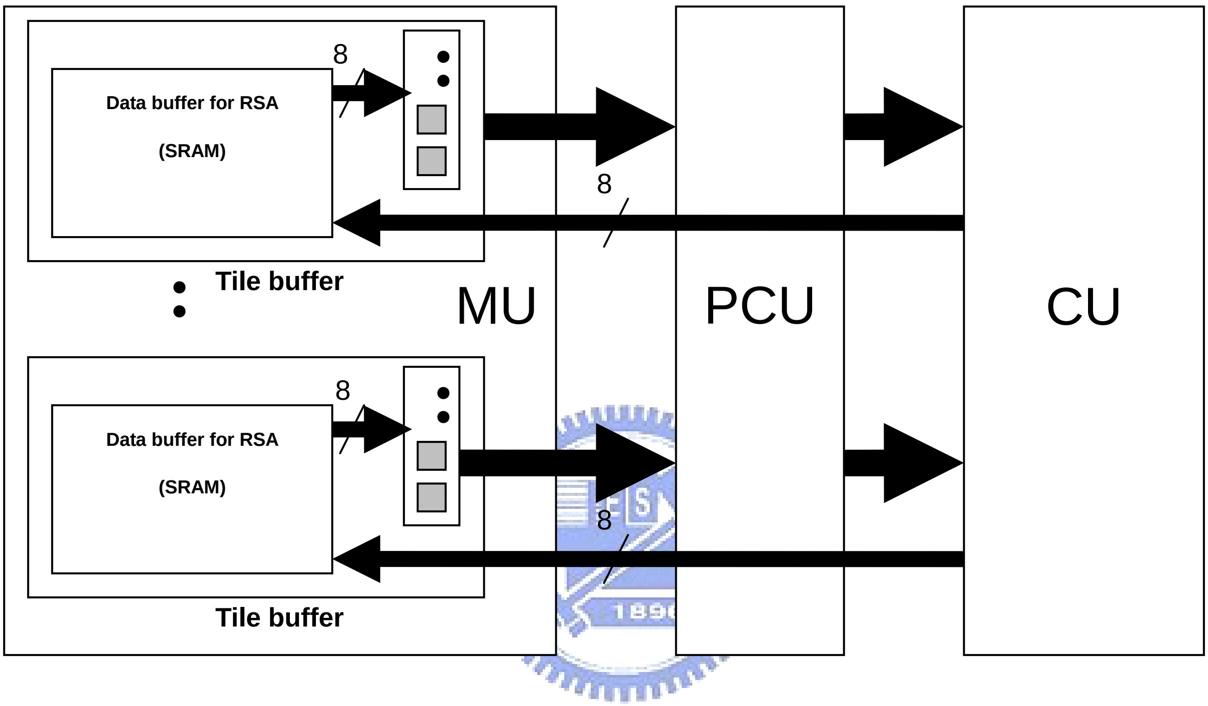


Fig. 3-24 Organization of MU

For table look up consideration, the organization of **MU** will be numbers of **Tile buffer** which shows in Fig. 3-24. Each Tile buffer has preload data buffer and large storage (SRAM).

3.6.2 Preload Buffer

The Preload Buffer is used to hide memory access in MON execution. As previous described, we partition variable length addition in some fix length addition. During the fix length additions switch, we need 4 byte data load and 1 byte data write back. Take the MON(A,B,N) as an example, the 4 byte load are N,B,B+N and previous accumulated result and the 1 byte write back is current result. If these memory accesses don't finish in one cycle, we will stall 5 cycles to perform these job. As a result, we design 4 byte preload data buffer during the MON execution. The preload buffer is shown in the following Fig. 3-25

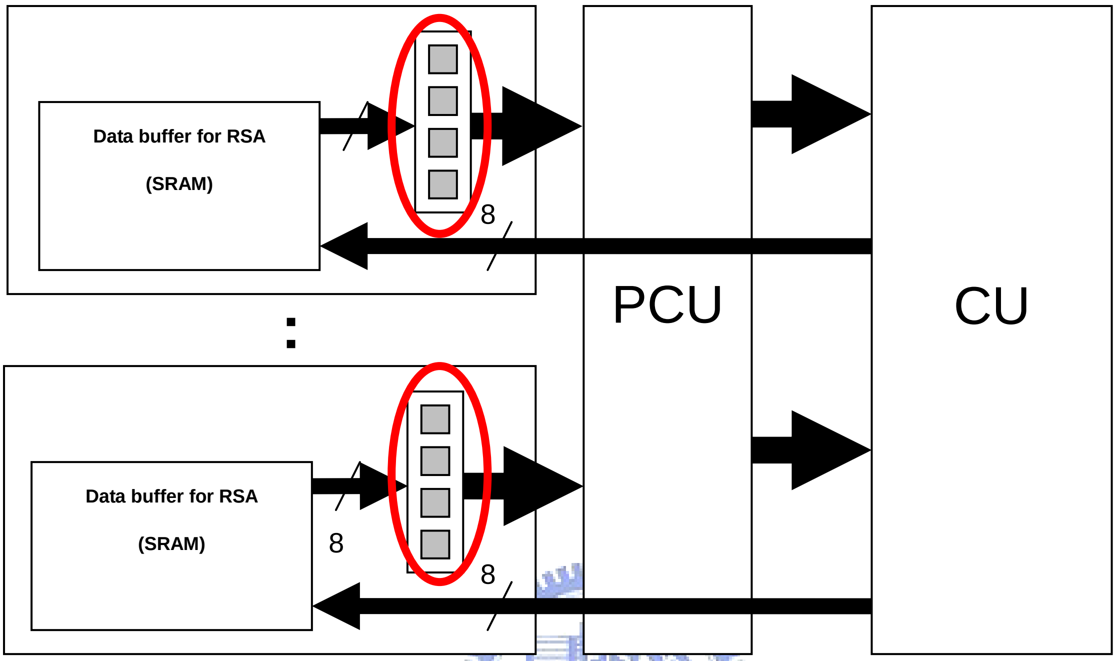


Fig. 3-25 preload buffer

3.6.3 Ratio of PE and Tile Buffer

The three cryptographic algorithms need different bandwidth requirements. Suppose that given ***m*** number of **PEs**. The memory bandwidth requirements for full utilization of **CU** and **MU** are shown in the Table 3-7. For AES , two memory accesses, that one is for **key scheduling** and another for **table look up,** are needed in one round. And our **PE** performs AES encryption and AES decryption in 3 cycles and 9 cycles respectively. For MON, the less **PE** number has, the more partition and memory access is needed. This is the reason why the memory bandwidth is an inverse proportion to **PE** number.

	AES		MON
Phase	Encryption	Decryption	En/Decryption
Bandwidth(bits/cycle)	m×2×8/3	m×2×8/9	7×8/m

Table 3-7 Bandwidth requirements for three cryptographic algorithms

As the table 3-7 shows, the bandwidth requirements of the three cryptographic algorithms are not the same. In order to get the best **space-time product,** we compute all **space-time product** in different ratio of **PE** number and **Tile Buffer** number and then we choose the most appropriate ratio. In the following Figures, M : N means M **PE**s : N Tile Buffers. And in Table 3-8 shows the area of **PE**, **Tile Buffer**.

	P.E.	Tile Buffer
Area	8216.208	71016.0290

Table 3-8 area of PE and Tile Buffer

AES

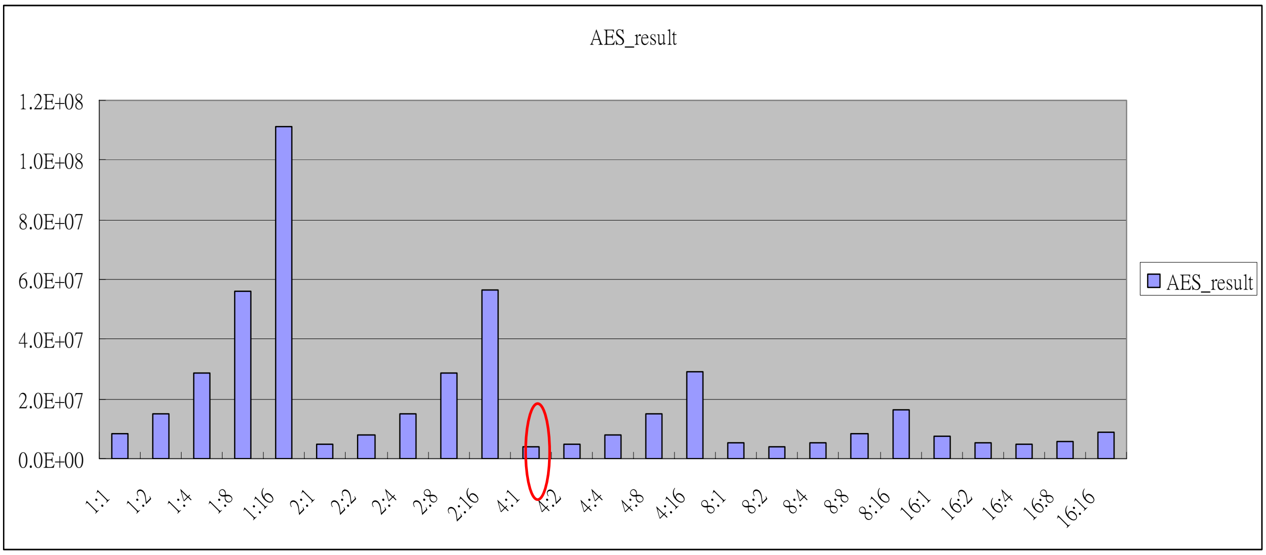


Fig. 3-26 Space-time product of different ratio in AES

The processing width of AES is 128-bit. It equals to 16 times of **PE** and **Tile Buffer.** The all ratio of **PE** and **Tile Buffer** under AES is from 1:16 to 16:1. The Fig. 3-26 shows **space-time product** of AES in all ratio. 4 **PE** and 1 **Tile Buffer** will get best result

Symmetric cryptography result

The Fig 3-27 shows that 4 **PEs** and 1 **Tile Buffer** will get best **space-time product.**

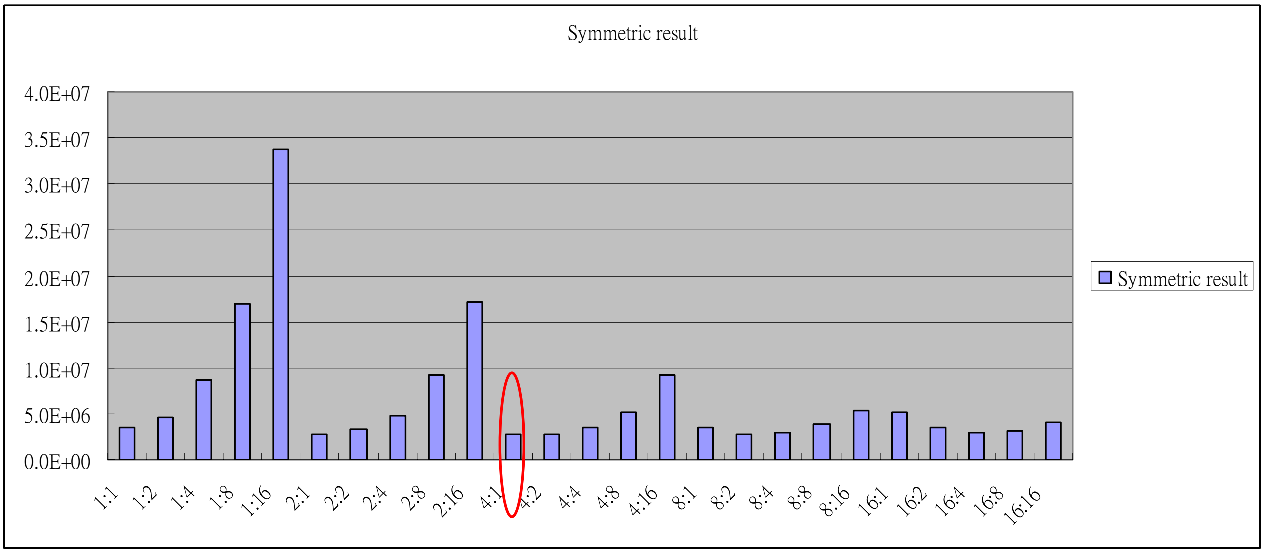


Fig. 3-27 Space-time product of different ratio in symmetric cryptography

Asymmetric cryptography result (MON)

We support MON from 256 ~ 4096. Nevertheless, as the Table 3-7 shows, 8 **Tile Buffer**s are enough. More than 8 **Tile Buffers** will cause hardware idle. As a result, we only consider **PE** from 1 to 16 and **Tile Buffer** from 1 to 8. In the Fig.3-28 the 16 **PEs** and 8 Tile Buffers will get best **space-time product.**

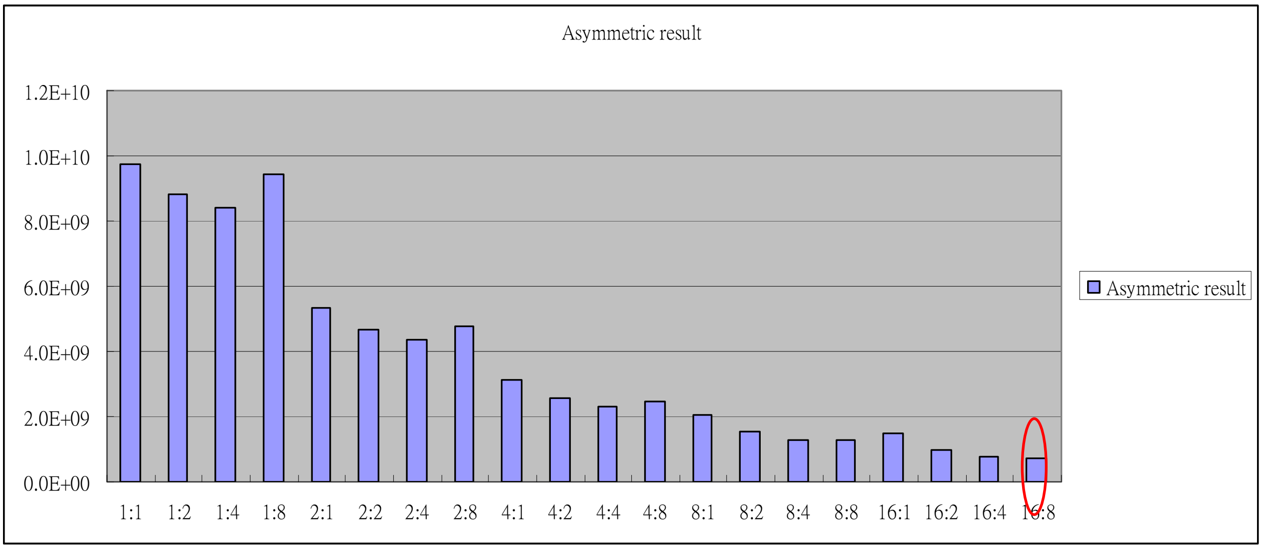


Fig. 3-28 Space-time product of asymmetric cryptography

Finial result

We combine AES and RSA in geometric mean. The result shows as follows. The best **PE** and **Tile Buffer** ratio is 16:8.

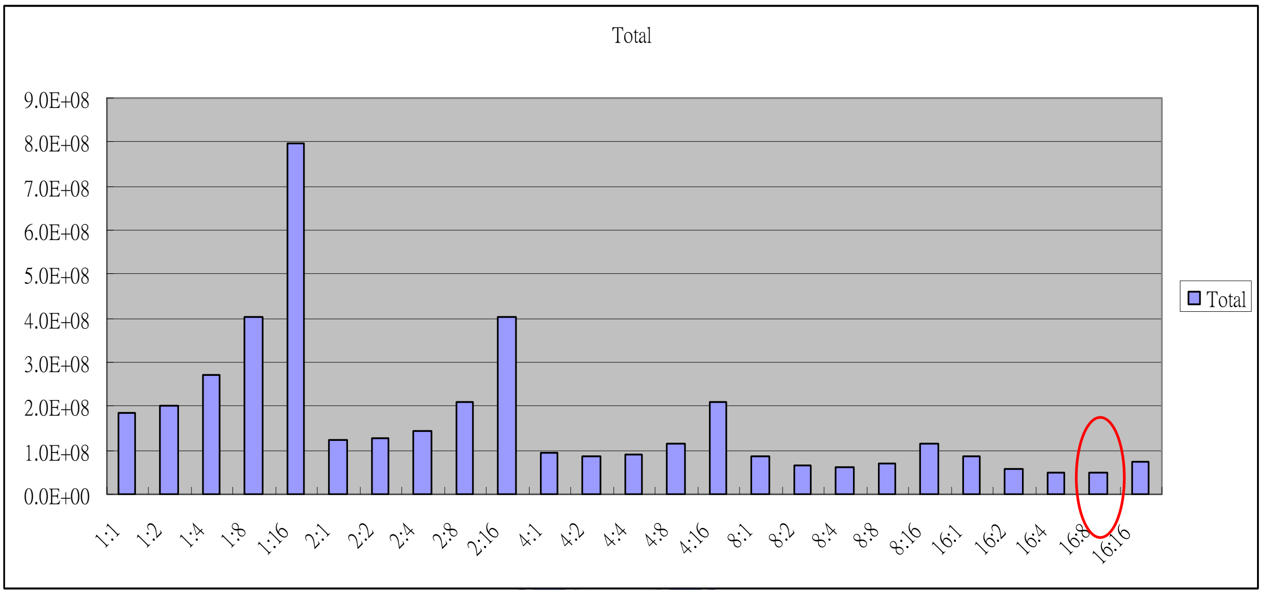


Fig. 3-29 Space-time product of total result

The Fig.3-30 shows that our system will be consisted of 16 **PE**s and 8 **Tile Buffers**.

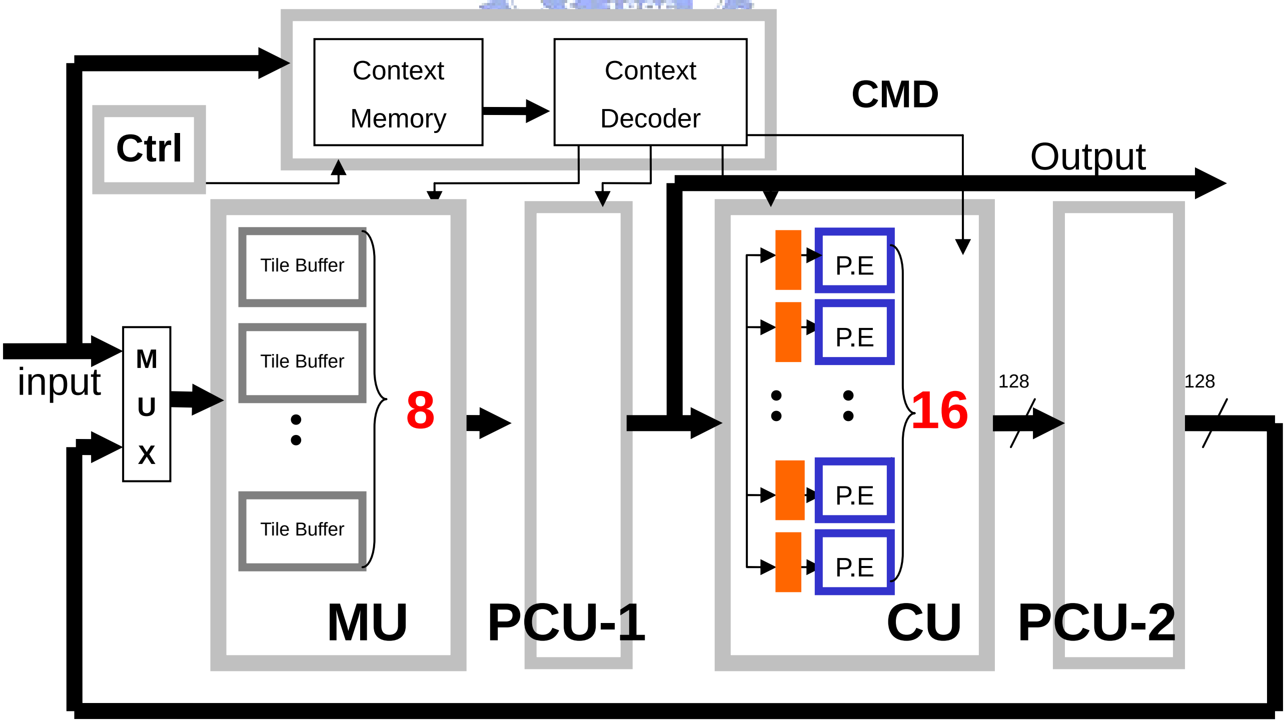


Fig. 3-30 System overview

3.7 Context Memory and Context Decoder

The three main components, **Permutation & Combination Unit**, **Computation Unit**, and **Memory Unit** have been designed. Their context is store in **Context Memory.** The **Context Memory** is reloadable. We can use new context sequency to perform new task or further cryptocraphy. **Context Decoder** decode context into control signal. Depending on three cryptographic algorithms, there two decode modes must be supported.

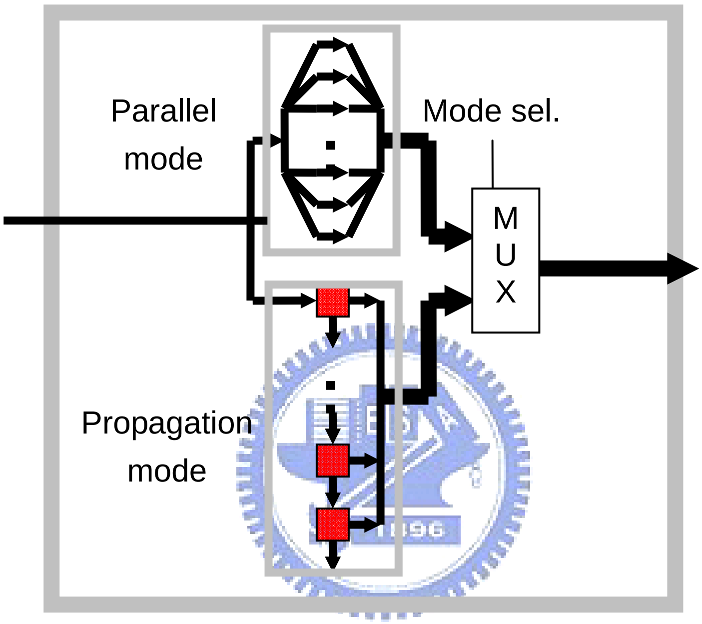


Fig. 3-31 Organization of Context Decoder

The organization of context decoder is shown as Fig. 3-31. Context Decoder has two modes to control CU and MU. Both of them are parallel mode and propagation mode.

Parallel mode

In this mode, all **PEs**, and **Tile Buffers** use the same control signal. This mode is for AES and DES.

Propagation mode

Because the carry propagates from the first **PE** to next **PE** during the MON execution, the control signal needs to reach **PE** and **Tile Buffer** in time to fit the carry propagation. In this mode, control signals are propagated from one **PE/Tile Buffer** to another. Under this mode, there are still two way for control P.E.

- Driven by context: Control signals are decode from the context. It is for usual addition.
- Driven by immediate data.: Control signals are depending on the computation results. This mode is only for MON.

3.8 CTRL

The **CM** and **CD** are used for control the three main part of our system. The **CTRL** is used to control the **CM** and **CM**. It controls how to load context in to **CM**, execution flow of context, and specify the Parallel or Propagation mode. This module is composed by the following component.

Address Generation Unit

This unit will produce address for **MU**. It composed by five base registers and six address generation function. By specify base registers and address function, we can generate address for **MU** to store and load data.

Control Flow Register

These register specify control flow of context. We support loop but not nested one. We need specify loop iteration number, loop step, loop start address, and loop end address during cryptographic algorithms execution.

A Finite State Machine

This finite state machine is used to control the whole hardware. The finite state machine is as Fig.3-31. It consisted of five states

- **System Idle** : This means that system is idle.

- **Flow Register Load** : This state will load flow registers, and address registers.
- **Context Memory Write** : This state means that write context to **CM**.
- **Execution Start** : This state means that our design will start to execution task

and indicate the external controller to put plaintext for en/decryption.

- **Task Execution** : In this state, execute the task that is specified by context.

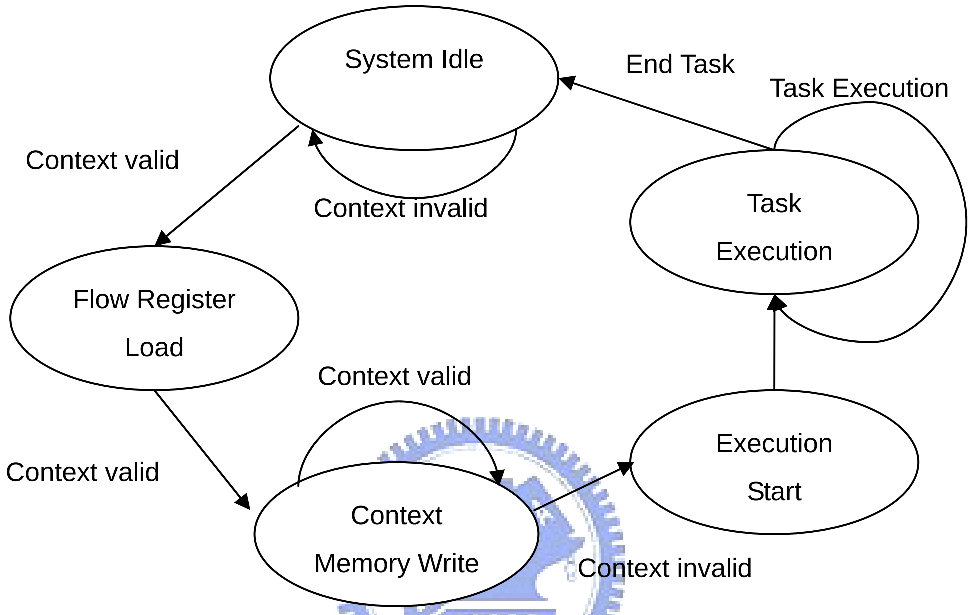


Fig. 3-32 Finite state machine of CTRL

Chapter 4 Evaluation Results

In this chapter, we first show the data rate which our design sustain. Then, we introduce ASIC designs for AES, DES and MON respectively. Thirdly, we compare the execution time and area between our design and ASIC. Last but not least, we compare the space-time product .Our design will get better results.

4.1 Evaluation Environment

First, we use the 0.18 μm library to synthesis our design and ASICs. And we use AES-128 and MON 1024 as benchmarks for AES and RSA.

Second, we can take off our SRAM in MU when our design put in some embedded system. We can use the system memory of embedded system. As a result, our design has two organizations: Stand along design and Take off SRAM design.

Last but not least, the evaluation metrics is shown as follows. S_0: space-time product of DES S_1: space-time product of AES A_0: space-time product of RSA. The performance metrics is $\sqrt{\sqrt{S_0 \times S_1} \times A_0}$ (2).

4.2 Processing Gap

The processing gap which is mentioned at section 1.1.1 is shown as follows.

Device	Cellular	Wireless Lan	Low-end network
Throughput (Mbits/sec)	0.128~2	2~60	100

Table 4-1 Processing Gap of secured embedded system

First, we use RSA-256 for key exchanging. And then, we use AES and DES to encrypt 1 M plaintext. The throughput is shown in the Fig. 4-1. The red line means data rate requirement of Wireless Lan and blue line for Low-end network. As the Fig.4-1 shows, our design can fit the processing requirement of cellular, Wireless Lan and low-end network.

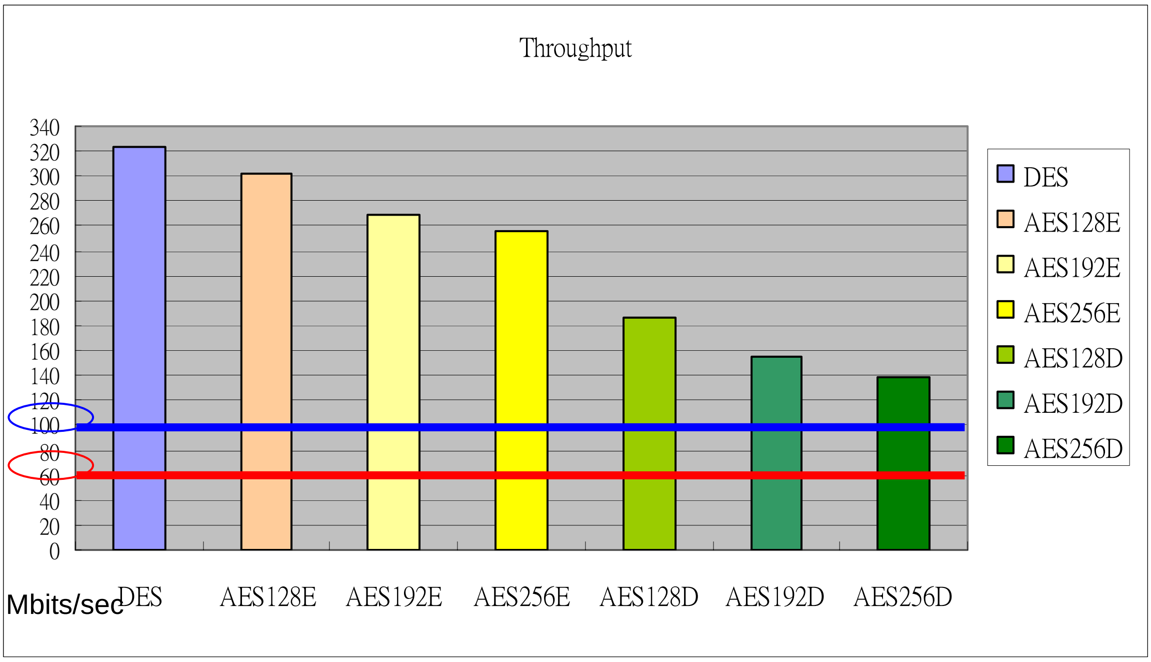


Fig. 4-1 Throughput of proposed design

4.3 ASIC

In this section, we introduce our ASIC design.

4.3.1 AES

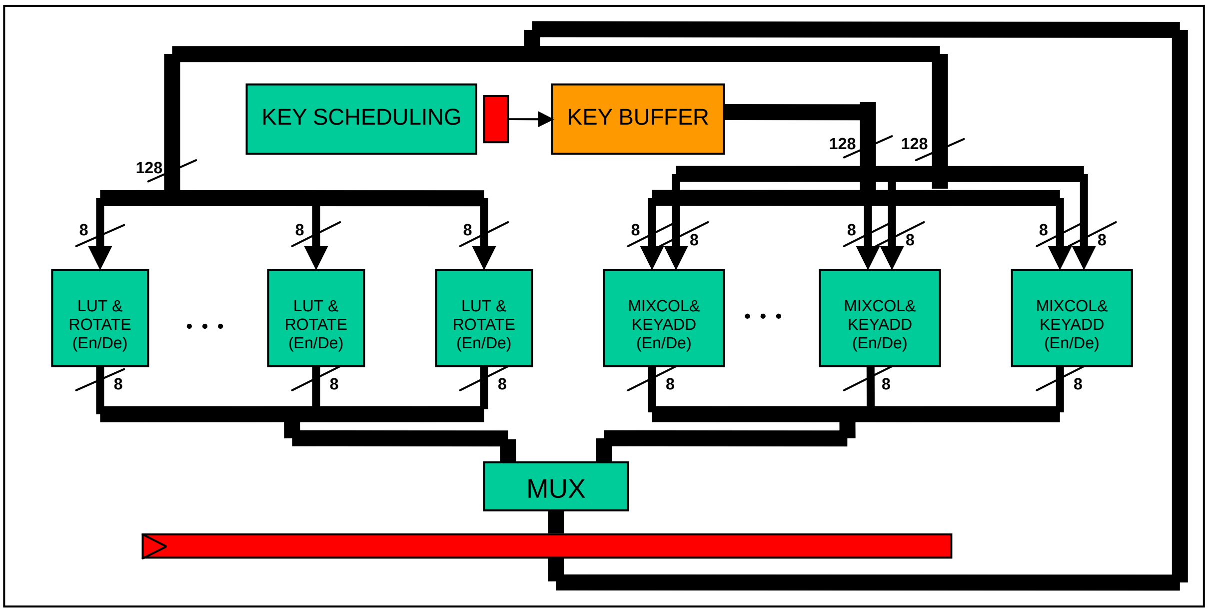


Fig. 4-2 ASIC of AES

Organization of AES ASIC is shown in Fig. 4-2. The width of data path is 128-bit. This ASIC have 4 components, Key Scheduling, Key Buffer, LUT&ROTATE and MIXCOL&KEYADD. The Key scheduling is used to generate key for encryption and decryption flow and then put it to Key Buffer. Key Buffer is a 64-bit SRAM.

LUT&ROTATE performs table look-up and bytes rotation. The Table look up is implemented by ROM. MIXCOL&KEYADD performs MixColumn and RoundKeyAddition. The encryption and decryption of AES in MixColumn are different. There are two parts in MixColum, one for encryption and another for decryption. Besides these two components, we still need one buffer for store immediate data.

One round of AES needs two cycles to execute. The area of each component is shown in Table 4-1 respectively. The cycle time of this design is 7.82 ns

Module name	LUT & ROTATE	MIXCOL & KEYADD	KEY SCHE.	KEY BUFFER	MISC.	TOTAL
Area ($10^{-6}mm^2$)	2×10^5	7×10^4	5×10^4	2.1×10^5	9×10^3	5.5×10^5
Timing(ns)						7.82

Table 4-2 Area or timing of each component in AES ASIC

4.3.2 DES

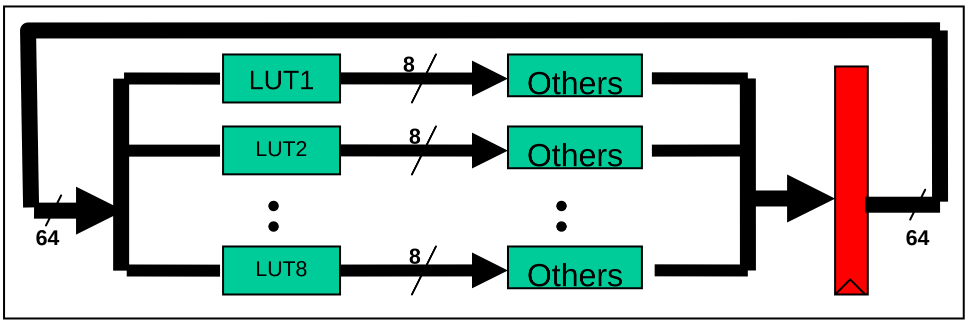


Fig. 4-3 ASIC of DES

DES ASIC design is shown in Fig 4-3. The width of data path is 64-bit. The look-up tables of DES are implemented by ROM. In DES, there are eight distinct tables. The others are consisted of some permutations and XOR. Table 4-2 is the detail of DES ASIC.

Module name	LUT	OTHERS	BUFFER	TOTAL
Area($10^{-6}mm^2$)	9×10^3	6.2×10^3	1.2×10^4	2.8×10^4
Timing(ns)				6.98

Table 4-3 Area or timing of each component in DES ASIC

4.3.3 MON

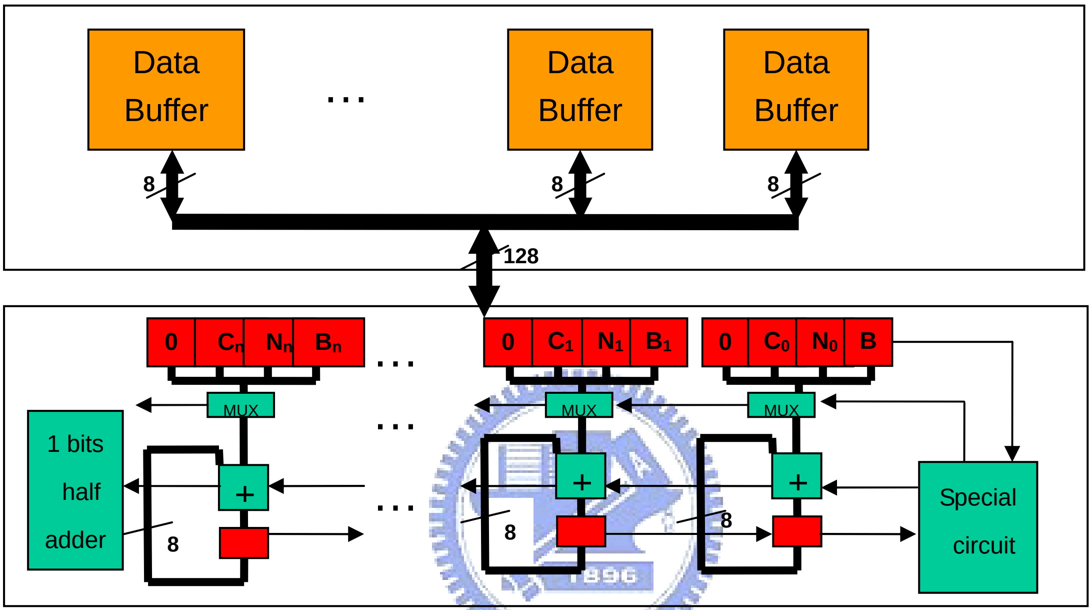


Fig. 4-4 ASIC of MON

The ASIC of RSA is shown as follows. This ASIC is designed to perform **Montgomery Algorithm**. It is similar with our system. It is consisted of 16 8-bit adders and 8 8-bit Data Buffer. As the analysis in section 3.5.3, this organization will get best **space-time product**. Table 4-4 is the ASIC detail of MON .

Module name	DATA BUFFER	Others	Total
Area($10^{-6}mm^2$)	5.8×10^5	1×10^5	6.8×10^5
Timing			5.06

Table 4-4 Area or timing of each component in RSA ASIC

4.3.4 ASIC Summery

The details of the three ASICs show as follows. Our hardware is designed to perform two 64-bit DES. In order to be comparable, the total area of ASIC is AES+RSA+DES*2. The cycle field in Table 4-5 means the cycle counts of 128-bit encryption or decryption for AES, DES and 1024-bit MON.

ASIC	AES_E	AES_D	DES	MON	Total
Area($10^{-6}mm^2$)	5.5×10^5		2.8×10^4	6.8×10^5	1.29×10^6
Timing(ns)	7.82		6.98	5.06	
Cycles	26	26	22	8334	

Table 4-5 Area timing and cycles of the three ASIC respectively

4.4 Proposed approach

In the following, it is our proposed approach. Its detail information shows in Table 4-8. Cycle counts shows in Table 4-9.

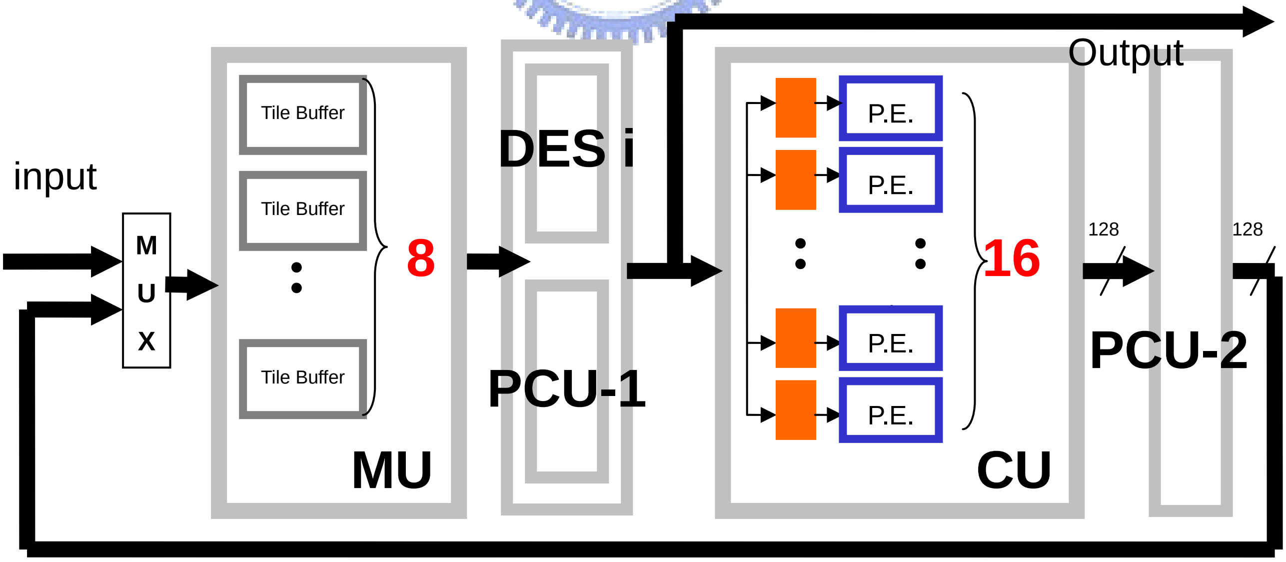


Fig. 4-5 Proposed approach

Module name	MU	PCU-1&DES	CU	PCU-2	Total
Area	5.8×10^5	9.8×10^4	1.4×10^5	1.8×10^4	8.6×10^5
Timing					5.46

Table 4-6 Area and timing of each component in modified approach

My Approach	AES_E	AES_D	DES	MON
Cycles	43	88	38	8334

Table 4-7 cycle counts of each cryptographic algorithm respectively

4.5 Timing and Area

Timing

Compare to ASIC design the performance loss is shows as follows. Our performance loss is just 33.38% in DES, 13.99% in AES encryption and 6.52% in MON. Unfortunately, the execution time of AES decryption is almost 2.3 times of ASIC. Because we choose finer-grain PE as our PE organization, the finer-grain PE will need more time than AES ASIC.

	DES	AES_E	AES_D	MON
Loss	133.38%	113.99%	233.28%	106.52%

Table 4-8 Performance loss

Area

Compare our design to dedicated design in the **stand alone design**. Our area need only 63.38% of ASIC area. In the Fig. 4-6, It shows the area of ASIC and our design.

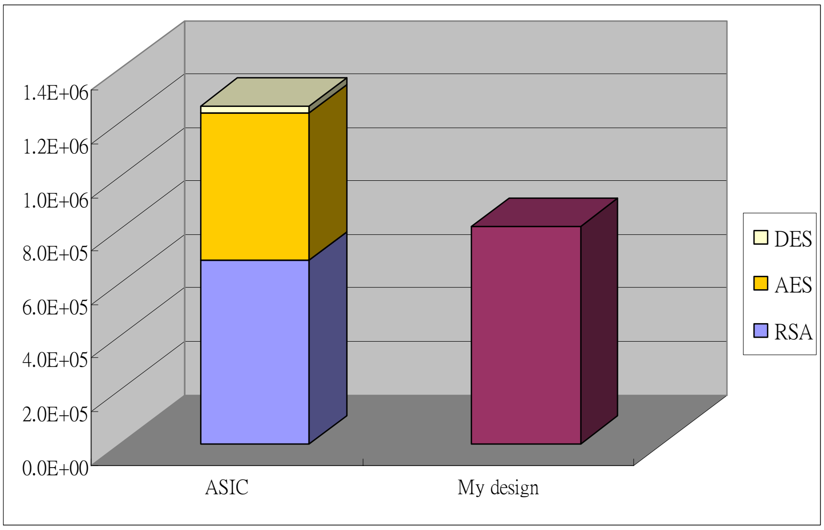


Fig. 4-6 Area result of stand along design

The comparison of Take off SRAM design is shown in Fig. 4-7. Our design need only 51.05% area of ASIC. The ASIC of MON can also replace it's SRAM with system memory. As a result, the MON ASIC will get less area than AES ASIC.

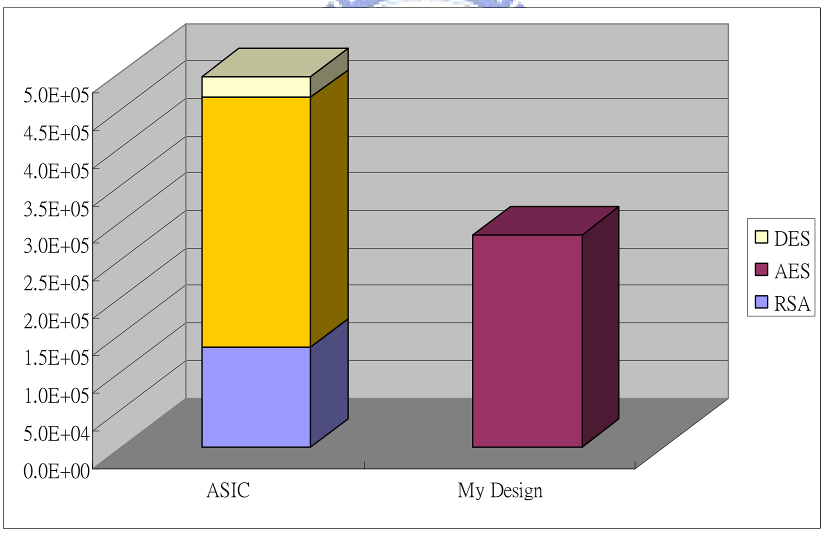


Fig. 4-7 Area result of stand along design

4.6 Space-time product

As previous described, our design can replace SRAM with system memory if we integrated in embedded systems. We assume that the memory bandwidth is 64-bit width. As a result, the throughput of take-off-SRAM design is the same with original one. In the following, we will show space-time product of this two types.

4.6.1 Stand alone design

The space-time product of stand alone design is shown as follows. The Fig. 4-8 is the result of symmetric cryptography. In DES, our design will get better space-time result than ASIC. However, AES will get worse result than ASIC because the execution time of AES decryption is 2.3 times more than ASIC. Fortunately, the result of symmetric algorithm is still better than ASIC result.

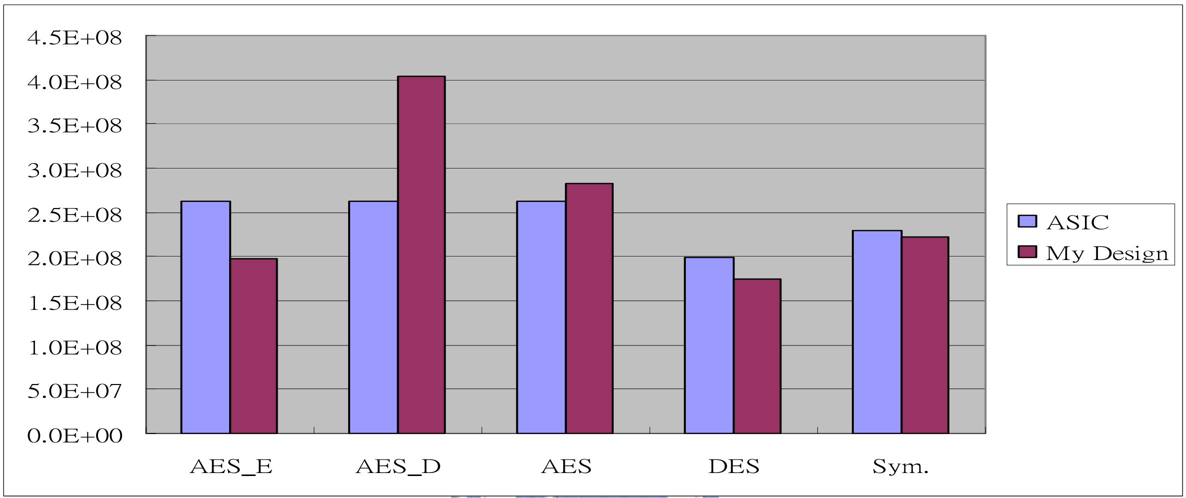
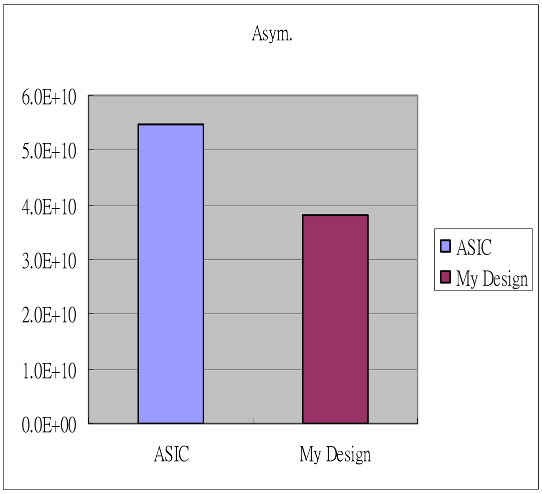


Fig. 4-8 Symmetric cryptography result

Result of asymmetric cryptograph is shown as Fig. 4-9. Our design will get better result than ASIC. Combine the result of symmetric and asymmetric cryptography. Our design will also get better space-time result than ASIC.



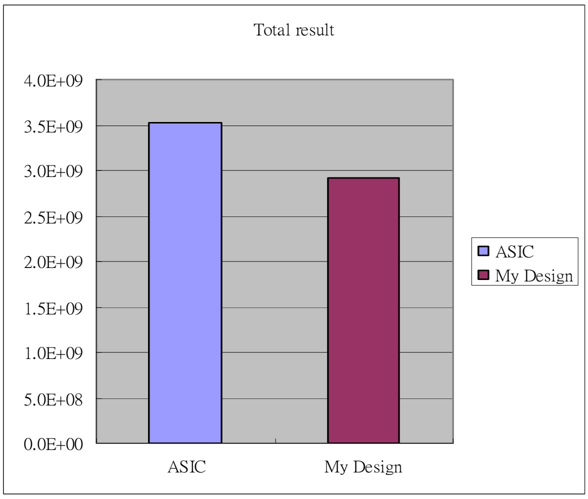


Fig. 4-9 Asymmetric cryptography result

Fig. 4-10 Result of sym. and asym. cryptography

4.6.2 Take off SRAM

The result of symmetric cryptography is shown as follows.

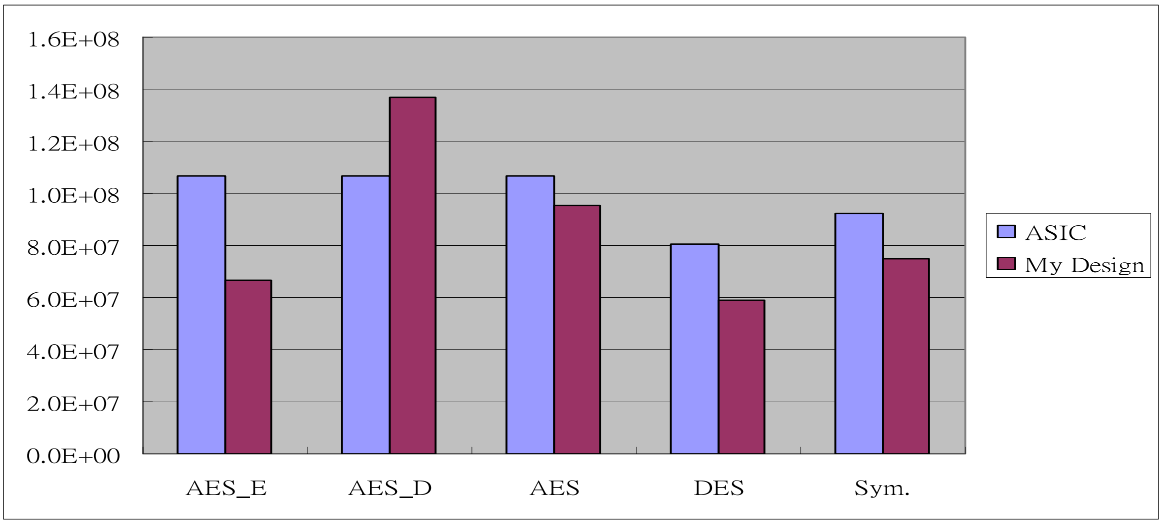


Fig. 4-11 Symmetric cryptography result

When we take off SRAM from our MU, the superiority of our design will be more obviously. The SRAM will need large area than others. But we can't optimize its area in cell base design.

The result of asymmetric cryptography is shown as follows. Our design will get better result than ASIC. Finally, combine the result of symmetric and asymmetric cryptograph. Our design will get better result than ASIC.

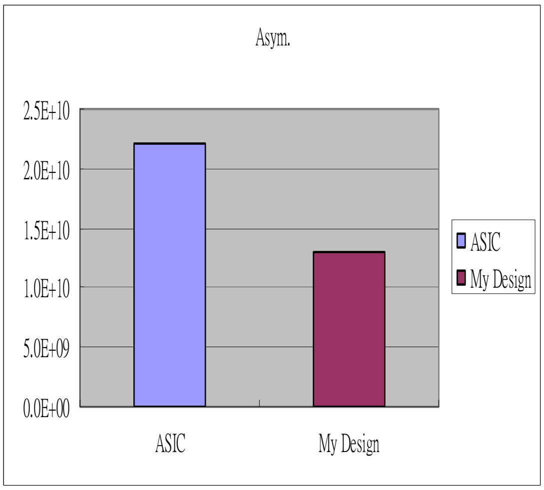


Fig. 4-12 Asymmetric cryptography result

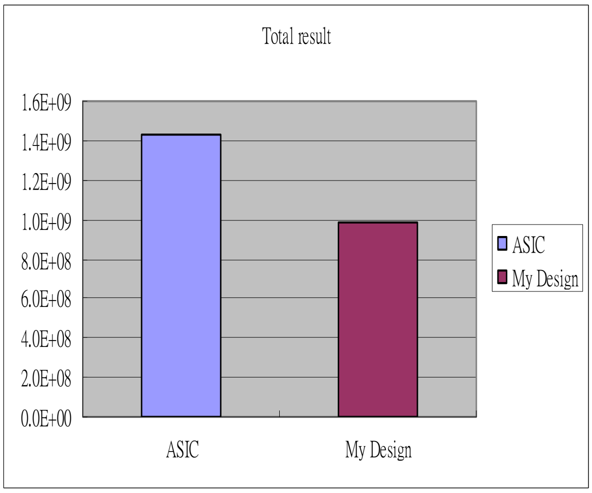


Fig. 4-13 Result of sym. and asym. cryptography

4.7 Summary

For stand alone system, our design needs 63.38% area of ASIC. For integration with embedded system, our design needs 51.05% of ASIC.

In performance, our system results just 6.52% performance loss of MON, 33.38% of DES, and 13.99% of AES Encryption. Only for AES decryption, our system may have less efficiency. It almost need 2.5 times of ASIC execution time.

Finally, the space & time product of our system will get better than ASIC design of these three.

Chapter 5 Conclusion and Future work

5.1 Conclusion

In this thesis, we have proposed a hardware which can switch flexibly between DES, AES, and RSA. And our design has significant area saving, but less performance loss. And the result of **space-time product** is also better than ASIC design.

Besides to **space-time product**, our system is a programmable engine with the most common operator, XOR and addition. Using different context sequence, we can perform new task composed by addition and XOR. For further cryptographic algorithms, we can apply them easily to our hardware.

In our design, the memory unit and computation unit both have high parallelism. The memory unit can perform 8 parallel table look up operation in one cycle and the computation unit can perform 16 parallel 8-bit additions. Our design will suit for some applications which need high parallelism. Take MPEG4 as an example. The motion estimation needs much high parallelism computation for pixels. In the quantification and inverse quantification, it also needs some parallel table look up.

Besides the high parallelism, our computation unit can perform very long addition in effective way. Some scientific application can also make use of the benefit to speed up computation.

5.2 Future Work

There are still some researches could be further studied. First, the PCU-1, and PCU-2 was dedicated for three cryptographic algorithms but its area is still up to 44% in total area in integrated vision. As a result, design a permutation unit with high efficiency and flexible is an important issue.

Second, we have proposed a methodology to decide what organization of **PE**. In order to avoid local optimal, this methodology choose several possible candidates and then make them into more detail evaluation. This flow makes us reduce problem space of finding **PE** organization. But our OPSET choosing method is just a structural method but doesn't provide global optimal. If the data flow graphs of the application are much larger, the method will waste lot of time and high complexity for graph computation. As a result, it needs a new method to decide possible candidates of P.E. organization with low complexity and have the ability for finding global optimal.

Reference

[1] http://www.epaynews.com/statistics/mcommstats.html

[2] Srivaths Ravi and Anand Raghunathan NEC Laboratories America Paul Kocher C ryptography Research and Sunil Hattangydy Texas Instruments Inc.. "Security in Embedded Systems: Design Challenges" ACM Transactions on Embedded Computing Systems, Vol. 3, No. 3, August 2004, Pages 461–491.

[3] http://java.sun.com/developer/technicalArticles/Security/Crypto/

[4] 近代密碼學及其應用 賴溪松，韓亮，張真誠 松岡

[5] http://big5.xinhuanet.com/gate/big5/news.xinhuanet.com/herald/2004-12/30/content_2394376.htm

[6] http://www-106.ibm.com/developerworks/library/s-crypt02.html

[7] W. Diffic and M. Hellman, "Privacy and Authentication: An Introduction to Cryptography." Proceedings of IEEE, 67 (1979), pp. 397-427.

[8] SPEC of DES : Federal information processing standard publication, "DATA ENCRYPTION STANDARD (DES)" October, 25, 1999.

[9] 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, Feb. 1978.

[10] E. Barker, L. Bassham, W. Burr, M. Dworkin, J. Foti, J. Nechvatal, and E. Roback,
"Report on the Development of the Advanced Encryption Standard (AES)." Available at
http://csrc.nist.gov/encryption/aes/round2/r2report.pdf

[11] SPEC of AES :
Joan Daemen, Vincent Rijmen , Document Version 2
"AES Proposal: Rijndael", Mar, 9, 1999.
Federal Information Processing Standards Publication 197 "Specification for the

ADVANCED ENCRYPTION STANDARD (AES)" November 26, 2001

[12] James F. Kurose keith W. Ross

"Computer Networking : A Top-Down ApproachFeaturing the Internet" 577~580

[13] P. L. Montgomery,

“Modular-multiplication without trial division,” Math. Comput.,

vol. 44, no. 7, pp. 519–521, 1985.

[14] Dino Oliva, Rainer Buchty, Nevin Heintze

"Embedded applications : AES and the cryptonite crypto processor"

Proceedings of the 2003 international conference on Compilers, architecture and synthesis for embedded systems, October 2003

[15] Henry Kuo, Ingrid Verbauwhede

"Architectural Optimization for a 1.82Gbits/sec VLSI Implementation of the AES Rijndael Algorithm"

Proceedings of the Third International Workshop on Cryptographic Hardware and Embedded Systems, May 2001

[16] Akashi Satoh, Sumio Morioka, Kohji Takano, Seiji Munetoh

"A Compact Rijndael Hardware Architecture with S-Box Optimization“

Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology December 2001

[17] Gaël Rouvroy, Francois-Xavier Standaert, Jean-Jacques Quisquater, Jean-Didier Legat

"Efficient Uses of FPGAs for Implementations of DES and Its Experimental Linear Cryptanalysis“

IEEE Transactions on Computers, Volume 52 Issue 4 April 2003

[18] Chih-Yuang Su; Shih-Am Hwang; Po-Song Chen; Cheng-Wen Wu

"An Improved Montgomery’s Algorithm for High-Speed RSA Public-Key Cryptosystem"

Very Large Scale Integration (VLSI) Systems, IEEE Transactions on
Volume 7, Issue 2, June 1999 Page(s):280 - 284

[19] Thomas Blum, Christof Paar
"High-Radix Montgomery Modular Exponentiation on Reconfigurable Hardware“
IEEE Transactions on Computers, Volume 50 Issue 7 July 2001

[20] Zhijie Shi and Ruby B. Lee,
“Bit Permutation Instructions for Accelerating Software
Cryptography”, Proceedings of the IEEE International Conference on
Application-Specific Systems, Architectures and Processors, pp. 138-148, July 2000

[21] Xiao Yang and Ruby B. Lee,
“Fast Subword Permutation Instructions Using Omega
and Flip Network Stages”, Proceedings of the International Conference on Computer
Design , pp. 15-22, September 2000

[22] Xiao Yang, Manish Vachharajani and Ruby B. Lee,
“Fast Subword Permutation Instructions Based on Butterfly Networks”,
Proceedings of Media Processors 1999
IS&T/SPIE Symposium on Electric Imaging: Science and Technology, pp. 80-86,
January 2000

[23] Ruby B. Lee, Zhijie Shi and Xiao Yang,
“How a Processor can Permute n bits in O(1) cycles,”,
Proceedings of Hot Chips 14 – A symposium on High Performance Chips,
August 2002

[24] Zhijie Shi, Xiao Yang and Ruby B. Lee,
“Arbitrary Bit Permutations in One or Two Cycles”,
Proceedings of the IEEE International Conference on Application-Specific
Systems, Architectures and Processors, June 2003

Appendix

A-1 Result of Integrated DES design

In this section, we show the space-time result which integrated DES with RSA and AES. In the Fig., A-1 is symmetric result which execute the AES encryption, AES decryption and DES. Fig. A-2 is asymmetric result which execute the MON. Fig. A-3 is the result which combines symmetric and asymmetric result.

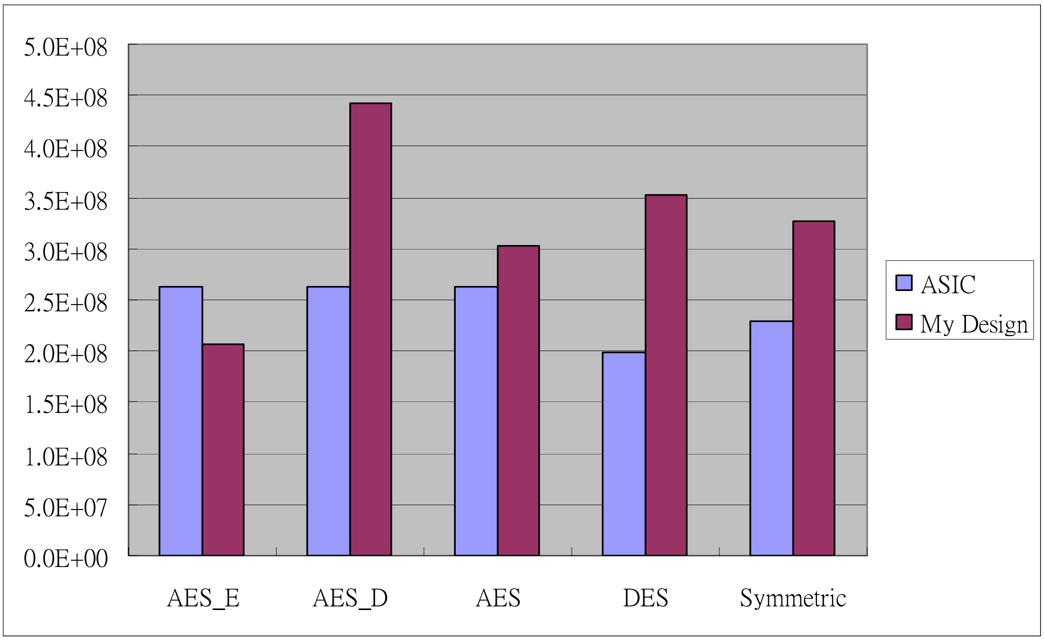


Fig. A-1 Symmetric result

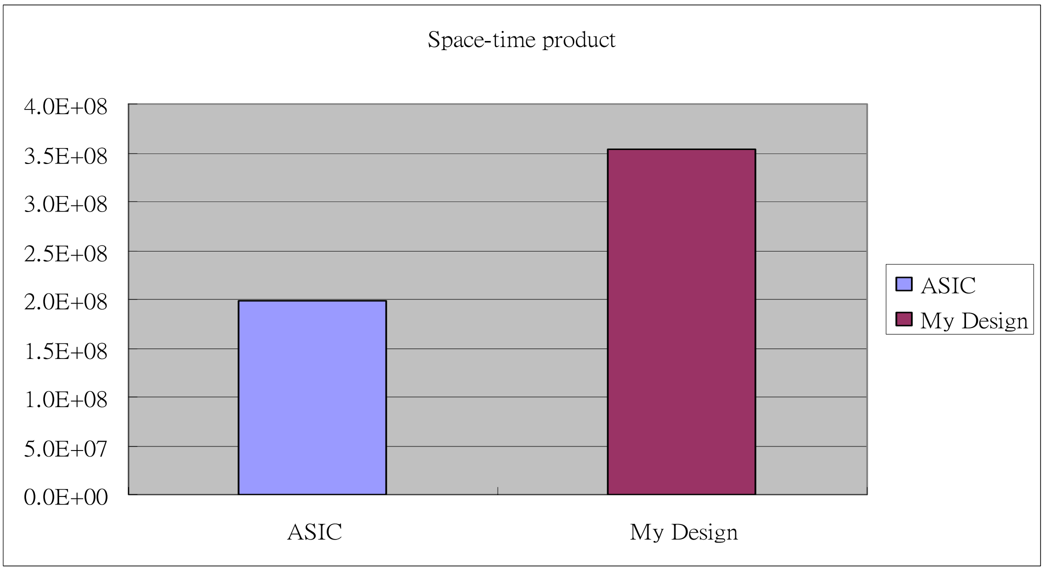


Fig. A-2 Asymmetric result

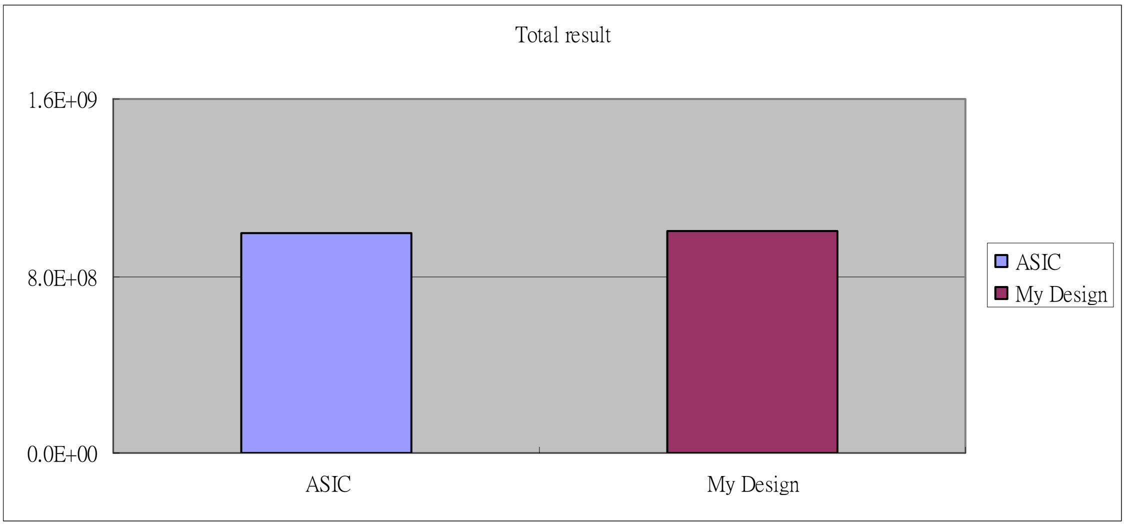


Fig. A-3 Total result

A-2 Related Product

AT91SC FAMILY: Atmel

Atmel's AT91SC Series of 32-bit RISC based secure microcontrollers provide the computing power and security levels required to meet the worldwide demand for next-generation applications.

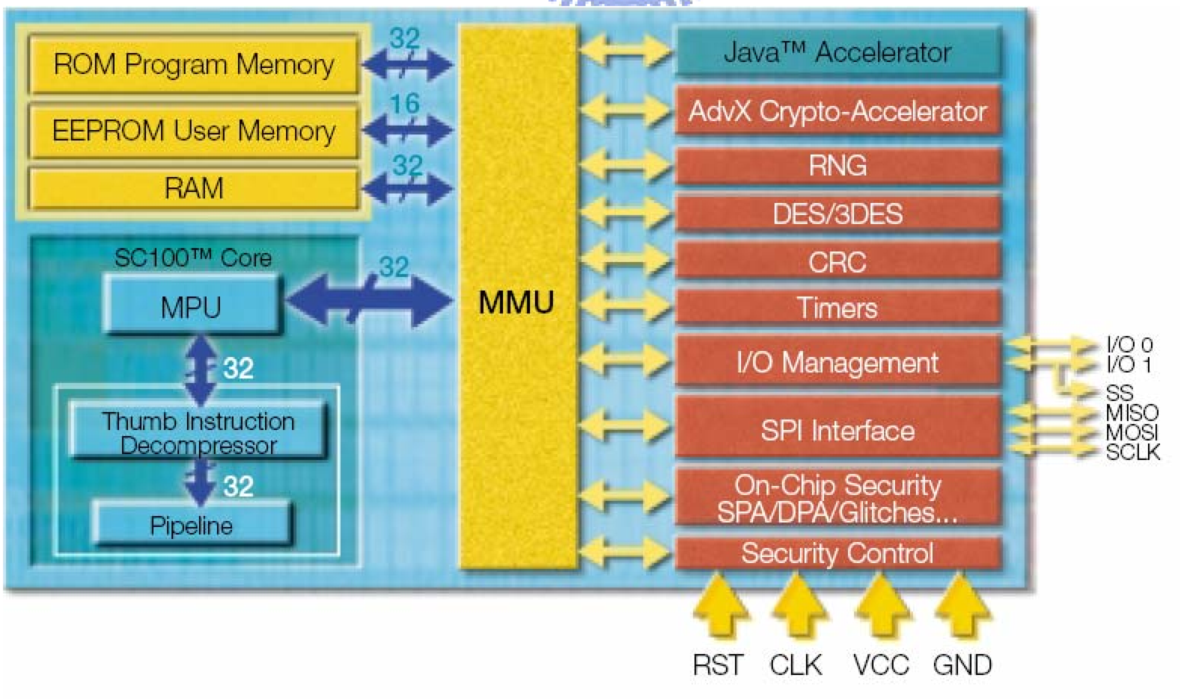


Fig. A-4 Organization of AT91SC

The following is its key feature and application example

Key Features

1. 32-bit RISC ARM® SecurCoreTM Up to 50 MHz Clock
2. JavaCard Hardware Accelerator
3. Advance Crypto Co-processor AdvXTM : RSA/DSA/ECC
4. Hardware DES and TDES
5. Advanced Interfaces : Two I/O Ports.
6. ISO 7816 Controller
7. SPI Interface
8. USB Full-Speed

Application Examples

1. SIM/USIM/UICC Cards
2. High Performance Smart Cards
3. Banking/IT/Pay TV, …
4. Secure Storage
5. Software Protection, e-token
6. Secure Access Module
7. High Security Applications