A Thesis for the Degree of Master

# A Study on Securing AES against Differential Power Analysis

Hwasun Chang

School of Engineering

Information and Communications University

2004

# A Study on Securing AES against Differential Power Analysis

# A Study on Securing AES against Differential Power Analysis

Advisor : Professor Kwangjo Kim

by

Hwasun Chang
School of Engineering
Information and Communications University

A thesis submitted to the faculty of Information and Communications University in partial fulfillment of the requirements for the degree of Master of Science in the School of Engineering

Daejeon, Korea
Dec. 29. 2003
Approved by

(signed)

Professor Kwangjo Kim
Major Advisor

# A Study on Securing AES against Differential Power Analysis

Hwasun Chang

We certify that this work has passed the scholastic standards required by the Information and Communications University as a thesis for the degree of Master

Dec. 29. 2003

Approved:

Chairman of the Committee
Kwangjo Kim, Professor
School of Engineering

Committee Member
Jae Choon Cha, Assistant Professor
School of Engineering

Committee Member
Daeyoung Kim, Assistant Professor
School of Engineering

M.S. Hwasun Chang
20022122

**A Study on Securing AES against Differential Power Analysis**

School of Engineering, 2004, 45p.
Major Advisor : Prof. Kwangjo Kim.
Text in English

# Abstract

Major credit card companies are planning to convert most of credit cards with magnetic stripe into smart cards within a few years. And usage of smart cards are increasing in such fields like transportation, electronic money, ID cards, etc. Major advantage of smart cards is that internal data like secret key can be used for internal processing and only the result is open to the public access.

However, the internal data kept inside smart cards and used internally can be found out using side channel attack. When cryptographic processing is occurred using input message and secret key, information like power consumption or electromagnetic radiation may be leaked. In side channel attack, the information is used to find out the secret key. Sometimes, attackers utilize timing information or induced faults during computation. Differential Power Analysis (DPA) is a kind of side channel attacks that makes use of power consumption information. DPA is a real threat because attackers can mount DPA with relatively cheap equipments and without knowing the internal implementation. Countermeasures against DPA can be divided into two categories. One is by hardware and the other is by software. Smart card chips manufactured

recently are equipped with hardware countermeasures. But it is generally recognized that DPA can be prevented effectively only by using both hardware and software countermeasures.

AES is the standard block cipher selected by NIST to replace DES in 2000. Masking methods were proposed as software countermeasure against DPA. But previous masking methods are vulnerable to Second Order DPA (SODPA) and can be made simpler in regard to memory and processing requirement.

In this thesis, simple fixed-value masking method that is resistant to SODPA and more efficient than previous methods is proposed and analyzed. The required memory for storing mask is 33% of previous method and the number of xor operation for applying mask is 18% of previous method. In practice, the reduction of memory usage will not affect the overall algorithm size much. But reducing the number of xor operations can improve the algorithm performance by about 10% in 32 bit smart cards optimized for speed. To prevent SODPA, we can make it hard for an attacker to catch the time when the mask is accessed and select mask for each round. In analysis process, the required properties of the generated masks, the appropriate number of masks, the required additional processing and memory for implementing the proposed countermeasure, and the security of the proposed method are suggested.

# Contents

# List of Tables

# List of Figures

# List of Abbreviations

**AES** Advanced Encryption Standard

**ALU** Arithmetic Logic Unit

**CPU** Central Processing Unit

**DES** Data Encryption Standard

**DPA** Differential Power Analysis

**EEPROM** Electrically Erasable Programmable Read Only Memory

**FIPS** Federal Information Processing Standard

**FVM** Fixed-Value Masking

**NIST** National Institute of Standards and Technology

**NSA** National Security Agency

**PODPA** Perhaps Optimal Differential Power Analysis

**RAM** Random Access Memory

**ROM** Read Only Memory

**SIM** Subscriber Identification Module

**SFVM** Simple Fixed-Value Masking

**SODPA** Second-Order Differential Power Analysis

**xor** exclusive-OR

# List of Notations

$a_{i,j}$ A byte in the State at $i$-th row and $j$-th column

$Ct$ Ciphertext

$f$ Intermediate value appearing in the way of monitoring that is function of plaintext (or ciphertext) and part of key

$FIN$ Mask applied before entering into $ByteSub$ transformation

$FK_{i,r}$ $r$-th fixed-mask for $i$-th round key

$FOUT$ Mask applied after leaving $ByteSub$ transformation

$n$ Block length of block cipher

$K^l$ $l$-th round key

$K'^{l,r}$ $l$-th round key masked with $r$-th fixed mask

$k^l_{i,j}$ A byte of $l$-th round key at $i$-th row and $j$-th column

$m_r$ Selected mask at position $r$

$Nb$ Number of columns of the State

$Nk$ Number of columns of secret key

$Nm$ Number of monitoring

$Nr$ Number of rounds

$P_i$ Constant for $x_i$ in linear power consumption model

$Pt$ Plaintext

$P(x)$ Power consumption of an instruction with operand $x$

$q$ Number of stored masks

$r$ Index for the selected mask and modified S-box

$S$ Original S-box used for $ByteSub$ transformation

$S'$ Modified S-box recomputed by the corresponding mask

$sk_{i,j}$ A byte in the secret key at $i$-th row and $j$-th column

$sm_r$ Subsidiary mask used in multiplicative masking

$t_{loadS'}$ The time when one byte of $S'$ output is loaded

$V_c(t)$ Power consumption trace of $c$-th monitoring with respect to time $t$

$x_i$ $i$-th bit of $x$

$\beta_d(x)$ Probability of $d$-th bit of $x$ being 0

$\delta 0_d(f)$ Set of $V_c(t)$s whose $d$-th bit of $f$ at the time of interest is 0

$\delta 1_d(f)$ Set of $V_c(t)$s whose $d$-th bit of $f$ at the time of interest is 1

$\Delta_d$ Differential power consumption curve, which is a differential of average power consumptions of $V_c(t) \in \delta 1_d(f)$ and $V_c(t) \in \delta 0_d(f)$

$\epsilon_{S'}$ Magnitude of the spike in differential power consumption for one byte output of $S'$

# I. Introduction

## 1.1 Block Cipher and Cryptanalysis

A block cipher is a function which maps $n$-bit plaintext to $n$-bit ciphertext block. $n$ is called the block length. The function is parameterized by a $k$-bit key $K$. It is generally assumed that the key is chosen at random. To allow unique decryption, the encryption function must be one-to-one. For $n$-bit plaintext and ciphertext blocks and a fixed key, the encryption function is a bijection. Each key potentially defines a different bijection. In block ciphers, same key is used for both encryption and decryption. On the other hand, asymmetric cryptosystem has different key for encryption and decryption.

DES is currently the most well-known block cipher developed by IBM. In 1977, DES was adopted by NIST as Federal Standard and afterwards DES was evaluated every 5 years to check if it is adequate as standard. In 1997, NIST decided that new algorithm is needed because attacks like exhaustive key search exploiting the short key length of DES had been demonstrated. Through three AES conferences, Rijndael was selected as AES in 2000 and became FIPS-approved block cipher that may be used by U.S. Government organizations and others to protect sensitive information.

Cryptanalysis is the study of techniques attempting to defeat cryptographic techniques. There are two kinds of cryptanalysis. One is software cryptanalysis and the other is hardware cryptanalysis. In software cryptanalysis, algorithm itself is analyzed and the examples are differential cryptanalysis and linear cryptanalysis.

One of hardware cryptanalysis is side channel attack. When cryptographic processing is occurred using input message and secret key, information like power consumption or electromagnetic radiation is leaked. In side channel attack, the information is used to find out the secret key. Sometimes attackers utilize timing information or induced faults during computation. DPA is a kind of side channel attacks that utilizes power consumption information.

## 1.2 Countermeasures for AES against DPA

A smart card is the card with microcontroller and looks like a credit card with yellow chip. In the microcontroller, there are CPU, serial communication interface, ROM, RAM, EEPROM, and cryptographic coprocessor. Until now, DES is the dominant block cipher implemented in smart cards. But DES will be replaced by AES before long because AES is more secure and efficient than DES and AES is the new standard. With the mentioned resources, a smart card can process input data and send the result through serial communication interface. Due to processing capability, the usage of smart cards is increasing. In the Asia Pacific region, Visa International is aiming to complete 90 percent migration of cards and terminals to smart cards by end of 2008. Smart cards are used also for electronic cash, royalty cards, ID cards and SIM of mobile phones. In those applications, smart cards are used as tamper-resistant devices. It is assumed that internal data like secret key are kept secretly and it is difficult for an attacker to know its real value.

But Kocher *et al.* [1] showed that the key used internally by smart cards can be found out by DPA. Because smart cards get power from outside and the structure is relatively simple, it is convenient to apply DPA on smart cards. After the announcement of DPA, many countermeasures have been proposed and used. Countermeasures can be

divided into hardware and software countermeasures. Many smart card chips manufactured recently have hardware countermeasures in them. And a new hardware countermeasure against side channel attack especially against probing attack was suggested in Crypto '03 conference [10]. But, Clavier *et al.* [3] showed that software countermeasures should be implemented even if hardware countermeasures exist. In real applications, many hardware and software countermeasures are used simultaneously to protect secret data in smart cards effectively.

As software countermeasure, Messerges [2] proposed random masking method for securing AES candidates against DPA. But this method uses random masks generated for each execution, and new S-boxes have to be computed accordingly. This requires more processing power and much RAM space is needed to store modified S-boxes. Because RAM is a limited resource in smart cards and processing power of smart cards is relatively low, this will not be a good solution especially for cheap smart cards.

To resolve the problem of random masking method, two kinds of improvements were proposed. One is multiplicative masking method and the other is fixed-value masking method. Multiplicative masking method was proposed by Akkar *et al.* [5]. But Golic *et al.* [6] showed that this method is inherently vulnerable to DPA. Some modifications of multiplicative masking method [6, 7] were proposed but they required much memory and processing.

Fixed-value masking method was suggested by Itoh *et al.* [4]. In this method, fixed masks and modified S-boxes are stored in ROM and a set of specific masks and modified S-boxes is selected randomly in the beginning of encryption. Because modified S-boxes are computed in advance and stored in ROM instead of RAM, it requires less processing and uses less RAM than random masking method. And because ROM has relatively more space than RAM in smart cards, this method is

practical.

## 1.3 Our Contribution

It seems that fixed-value masking method is the most suitable software countermeasure for securing AES against DPA especially in cheap smart cards. But existing fixed-value masking method is vulnerable to SODPA [8]. SODPA uses two points in a power consumption curve and it can be used for breaking masking method. And existing masking method has room for improvement in respect to memory usage and processing requirement. In this thesis, we propose new method which is characterized by memory-saving and efficiency and resistant to SODPA as well.

## 1.4 Outline of the Thesis

The remainder of the thesis is organized as follows. In Chapter II, some underlying concepts and primitives are explained. In Chapter III, existing masking methods are described. In Chapter IV, the improved masking method is proposed and analyzed in regard to security and performance in Chapter V. In Chapter VI, the proposed method is compared with the previous method. Finally, we end with concluding remarks in Chapter VII.

# II. Preliminaries

## 2.1 AES

AES consists of an initial round key addition, variable $Nr - 1$ rounds and a final round. In this section, we brief some parts of AES that is related with thesis.

### 2.1.1 State, Secret Key, and Number of Rounds

The different transformations operate on the intermediate result called "State". The State can be pictured as a rectangular array of bytes. This array has four rows. The number of columns is denoted by $Nb$ and is equal to four. The secret key is similarly pictured as a rectangular array with four rows. The number of columns of the secret key is denoted by $Nk$ and equal to the key length divided by 32. $Nk$ can be 4, 6, or 8. These representations are illustrated in Fig.2.1.

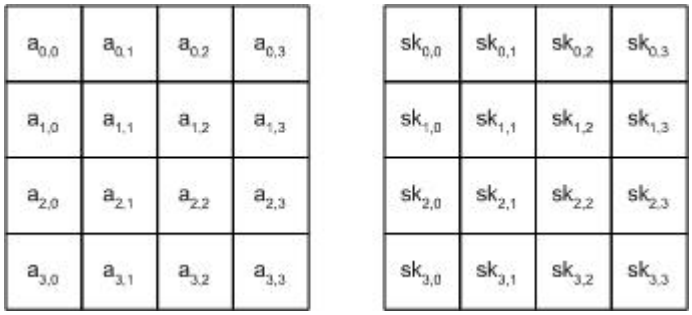


Figure 2.1: State (with $Nb = 4$) and secret key (with $Nk = 4$)

Table 2.1: Number of rounds ($Nr$)

| $Nr$ | $Nk = 4$ | $Nk = 6$ | $Nk = 8$ |
|---|---|---|---|
| $Nb = 4$ | 10 | 12 | 14 |

The number of rounds denoted by $Nr$ can be fixed by the value $Nk$, which is given in **Table 2.1**.

### 2.1.2 Round Transformation

The round transformation is composed of four different transformations which can be represented in pseudo C notation as follows:

$Round(State, RoundKey)\{$
    $ByteSub(State);$
    $ShiftRow(State);$
    $MixColumn(State);$
    $AddRoundKey(State, RoundKey);$
$\}$

In the final round, $MixColumn$ is omitted.

#### ByteSub transformation

$ByteSub$ transformation is a non-linear byte substitution operating on each byte of the State independently. It can be processed by table loop up and the table is called S-box.

#### ShiftRow transformation

In $ShiftRow$ transformation, the rows of the State are cyclically shifted over different offsets. Row zero is not shifted, row one is shifted over

one byte, row two is shifted by two bytes and row three is shifted over three bytes.

**MixColumn transformation**

In $MixColumn$ transformation, the columns of the State are considered as polynomials over $GF(2^8)$ and multiplied modulo $x^4 + 1$ with a fixed polynomial $c(x)$ given by $c(x) = \text{`}03'x^3 + \text{`}01'x^2 + \text{`}01'x + \text{`}02'$. This can be written as a matrix multiplication.

$$\begin{bmatrix} b_{0,j} \\ b_{1,j} \\ b_{2,j} \\ b_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_{0,j} \\ a_{1,j} \\ a_{2,j} \\ a_{3,j} \end{bmatrix}$$

**AddRoundKey transformation**

In $AddRoundKey$ transformation, round key is applied to the State by a simple bitwise xor.

### 2.1.3 Key Schedule

Round keys are derived from secret key by means of the key schedule. This consists of key expansion and round key selection. The expanded key is a linear array of four byte words and is denoted by $W[Nb * (Nr + 1)]$. The first $Nk$ words contain the secret key. If $Nk$ is equal to or below 6, key expansion can be represented by the following pseudo C notation.

$$\begin{aligned} &KeyExpansion(byte\ Key[4 * Nk],\ word\ W[Nb * (Nr + 1)])\{ \\ &\quad for(i = 0; i < Nk; i++) \\ &\qquad W[i] = (Key[4*i], Key[4*i+1], Key[4*i+2], Key[4*i+3]); \\ &\quad for(i = Nk; i < Nb * (Nr + 1); i++)\{ \end{aligned}$$

$temp = W[i-1];$
$if(i\%Nk == 0)$
$temp = SubByte(RotByte(temp)) \oplus Rcon[i/Nk];$
$W[i] = W[i-Nk] \oplus temp;$
}
}

## 2.2 DPA

### 2.2.1 Power Consumption Model

Akkar *et al.* [12] derived power consumption model of two kinds of smart cards by experiment. High-end smart cards could be represented by linear model. When operand $x$ is manipulated using assembly command like *store* or *load*, the corresponding power consumption $P(x)$ can be represented by the following equation.

$$P(x) = \sum_{i=0}^{u-1} x_i P_i$$

In the equation, $u$ is the number of bits in $x$ and $x_i$ represents the $i$-th bit of $x$. $P_i$ is a constant given by the experiment. $P_i$ is not a constant value and has plus value for some $i$ and negative value for the others. From this result, we can see that Hamming weight model which states that power consumption is proportional to the number of 1's in operand, is not always correct.

Although low-end smart cards cannot be represented by the linear model, two kinds of cards had a common power consumption property. Average power consumption was different when $x_i$ was zero and when $x_i$ was one. And the average power consumption with $x_i = 1$ was not always greater than the average power consumption with $x_i = 0$.

### 2.2.2 DPA

In DPA, power consumption information is analyzed statistically to find out the secret key. First, the attacker select a bit that is function of plaintext and part of key. For the selected bit, usually a bit of S-box output is selected. The result of $AddRoundKey$ transformation can be used, too. By recursively executing the attack procedure, an attacker can find out the part of key. For the part of key, attacker should try with all combinations of the value in the worst case. But because the attacker can find out the secret key part by part, the required number of execution is much less than the exhaustive search for the entire key. For example, if the attacker finds out the secret key 8 bits at a time, the attacker should try $2^8$ in the worst case to find out the part of key. If the key size is 128 bits, an attacker can find out the secret key by trying $2^8 * 16 = 2^{12}$ times in the worst case. To find out the secret key with 128 bits using exhaustive search, the attacker should try $2^{128}$ times in worst case. From this, we can see that the number of trials is reduced dramatically using DPA.

### 2.2.3 DPA on AES

In this part, detailed procedure for applying DPA on AES is described. As stated in the previous section, DPA can be applied at the output of S-box or at the result of round key addition. Concrete procedures for two approaches are as follows:

#### DPA on S-box Output

This method was proposed by Kocher *et al.* [1] for attacking S-box of DES. From each execution of the following procedure, an attacker can find out eight bits of the first round key. By executing this sixteen times, the attacker can find out the entire first round key. In most cases, the

length of the secret key for AES is 128 bits and the first round key is same with the secret key. If the length of secret key is longer than 128 bits, the second round key contains the remaining part of the secret key. The second round key can be found out from the second *ByteSub* transformation using similar procedure as the following and the first round key. The procedure for attacking the first eight bits of the first round key is like this.

DS1. The attacker generates many plaintexts randomly.

DS2. Encryption is done with the plaintexts using the unknown secret key and the corresponding power consumption curves are recorded.

DS3. The attacker selects a bit of first *ByteSub* transformation output, which is a function of first byte of plaintext and first byte of first round key.

DS4. The attacker assumes a value for the first byte of the first round key, and from the assumed part of the key and plaintext, an attacker calculates the value of the selected bit.

DS5. According to the value of the selected bit, power consumption curves are divided into two groups.

DS6. The attacker calculates the average power consumption curve of each group and the differential power consumption curve of two average curves. If the key assumption is correct and accordingly grouping is correct, two average power consumption curves will show difference when the selected bit is manipulated. At that point, we can observe a spike in differential power curve.

### DPA on the Result of Round Key Addition

This method was proposed by Chari *et al.* [24] to attack the whitening process of Twofish encryption algorithm. Whitening is the technique for extending the length of key without modifying algorithm. For whitening, key is xor-ed with data before the algorithm and after the algorithm. In the attack, the fact that bit value of one in key inverts the corresponding bit value of data is utilized. If the bit value of key is zero, the corresponding data bit remains unchanged after xor operation.

If we divide the power consumption curves according to the value of the bit value of data that is xor-ed with the key bit, we will obtain two groups of curves. After calculating average curve of each group, we can get differential curve of two average curves. When the key bit is zero, we will observe two spikes with same direction in the differential curve. This is because the difference of data bit exists in two points of the differential curve and the bit value of data has not been changed.

If the bit value of key is one, the corresponding bit value of data has different value before and after being xor-ed with key. If we divide the power consumption curves according to the value of the data bit value that is xor-ed with the key bit, the differential curve of average power consumptions will have two spikes with opposite direction. This is because the bit value of data has been changed.

By attacking $AddRoundKey$ transformation, an attacker can find out the first and second round keys. Therefore, as explained in the previous part, an attacker can find out the secret key. The concrete procedure for finding out a bit of the first round key is as follows:

DK1. The attacker generates many plaintexts randomly.

DK2. Encryption is done with the plaintexts using the unknown secret key and the corresponding power consumption curves are recorded.

DK3. The attacker selects a bit of the first round key. And the attacker divides pairs of plaintext and the corresponding power consumption curve into two groups according to the bit value of data that is xor-ed with the selected round key.

DK4. The attacker calculates the average power consumption curve for each group and the differential curve of two average curves. If the key bit value is one, the attacker can observe two spikes with opposite direction in the differential curve. Otherwise, the attacker can observe two spikes with same direction in the differential curve.

# III. Ways of Masking

## 3.1 Random Masking

Messerges [2] proposed random masking for securing AES candidates against DPA. He suggests modified primitive operations so that masks are applied. The modified primitives are as follows:

### Table Loopup

A table lookup operation $S$ that has $x$ as input and $y$ as output is symbolically denoted as $y = S[x]$. When input mask $m_{in}$ and output mask $m_{out}$ are used, masked table $S'$ can be defined by the following equation.

$$S'[x] = S[x \oplus m_{in}] \oplus m_{out}$$

The masked table $S'$ takes inputs that are masked with $m_{in}$ and produces outputs that are masked with $m_{out}$.

### Bitwise Boolean Function

To compute the xor-ed value of two masked operands, compute xor-ed value of the masked operands and xor-ed value of their corresponding masks. If the masked operands $x'$ and $y'$ are masked with $m_x$ and $m_y$, respectively, the masked output is $z' = x' \oplus y'$ and the new mask is $m_z = m_x \oplus m_y$.

### Shift Operation

In this operation, the masks simply shift along with the data.

**Polynomial Multiplication over $GF(2^8)$**

When the multiplication is performed using table lookups, shift and xor operations, the corresponding methods to protect these operations can be used.

Using the above modified primitives, masked data are encrypted with masked key and the corresponding masks should be calculated at the same time. At the end, masks are applied to get the unmasked ciphertext. For this method, new masks should be generated randomly for each byte in the State and for each round. But this method requires much processing and memory. Each encryption should be accompanied by the mask calculation. And for each mask, new table for *ByteSub* transformation should be constructed temporarily. Messerges commented the possibility of using the same mask for different variables and rounds and suggested use of fixed mask. But he did not mention the detailed method.

## 3.2 Multiplicative Masking

Main problem of implementing a masking method comes from the non-linear parts of the algorithm due to S-box recomputation, which requires much processing and RAM.

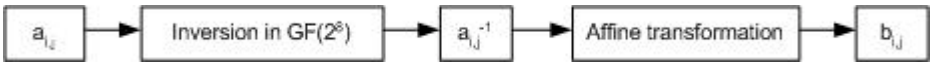


Figure 3.1: *ByteSub* transformation

To resolve the problem, Akkar *et al.* [5] proposed multiplicative masking. For AES, boolean mask is transformed into multiplicative

mask before inversion operation because *ByteSub* transformation is composed of inversion in $GF(2^8)$ and affine transformation as shown in Fig.3.1. When a byte in the State $a_{i,j}$ is masked with mask $m_r$, we can get the same mask after inversion using the scheme shown in Fig.3.2. With the scheme, S-box need not be recomputed and stored.

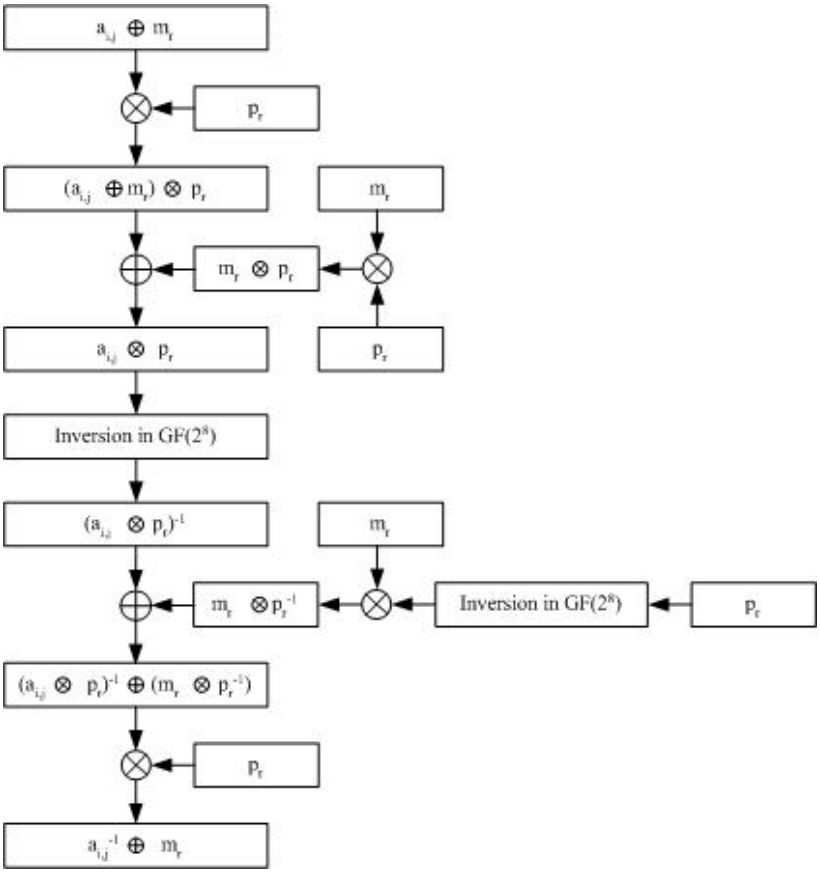


Figure 3.2: Modified inversion with multiplicative masking

But Golic *et al.* [6] showed that multiplicative masking is vulnerable to DPA. The basic problem with the multiplicative masking is that it does not mask the all-zero byte value of data.

## 3.3 Fixed-Value Masking

In this method, $q$ sets of masks and the corresponding modified S-boxes are computed in advance and stored in ROM. Two types of fixed-value masking methods were proposed by Itoh *et al.* [4]. In the first method, the applied masks are same across rounds but they are different from byte to byte in the State. In the second method, the masks are same across rounds and bytes in the State.

When the encryption is executed, a set of masks and modified S-boxes are selected randomly and used. Because second method uses less ROM, it is more adequate for cheap smart cards. AES algorithm using the second method is shown in Fig. 3.3 and **Algorithm 1**. **Algorithm 2** is the modified version of $ByteSub$ transformation so that modified S-box stored in ROM can be used. $ShiftRow$ and $MixColumn$ are the same with the original transformations.

There are three kinds of masks $FK_{i,r}$ used for masking round keys. That is for $i = 0$, $i = 1, \ldots, Nr-1$ , and $i = Nr$ where $Nr$ is the number of rounds in AES. And these masks are derived from FIN and FOUT. FIN is the mask that the bytes in the State have before entering into $ByteSub\_FM$ transformation and FOUT is the mask that the bytes in the State have after $ByteSub\_FM$ transformation. FIN and FOUT are generated randomly so that any bit of the masks is 0 with probability 1/2. Equation (3.3) shows how $FK_{i,r}$ is derived from FIN and FOUT. Modified S-boxes are computed by **Algorithm 3**.

**Algorithm 1** $FixedMasking_{AESEnc}$

$FixedMasking_{AESEnc}(Pt)$
1 $\quad /* \ K'^{\,i,r} : masked\ round\ key\ (i = 0, \ldots, Nr, r = 0, \ldots, q-1) \ */$
2 $\quad r = GenerateRandomNumber(); \ /* \ choose\ r = 0, \ldots, q-1 \ */$

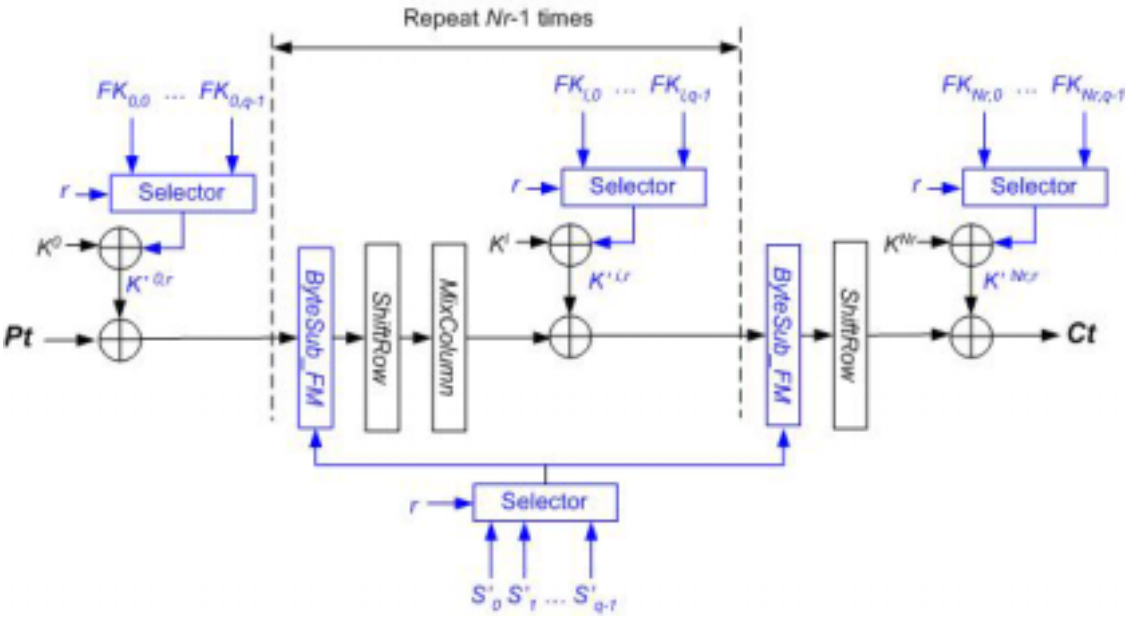


Figure 3.3: Fixed-value masking

3 $T' = Pt;$

4 $for(i = 0; i < Nr - 1; i++)\{$

5 $\quad T' = T' \oplus K'^{\,i,r};$

6 $\quad T' = ByteSub\_FM(T', r);$

7 $\quad T' = ShiftRow(T');$

8 $\quad T' = MixColumn(T');$

9 $\}$

10 $T' = T' \oplus K'^{\,Nr-1,r};$

11 $T' = ByteSub\_FM(T', r);$

12 $T' = ShiftRow(T');$

13 $T = T' \oplus K'^{\,Nr,r};$

14 $output\ T;$

In **Algorithm 1**, $T'$ is the intermediate masked value. Let $K^i$ be the original round key and $FK_{i,r}$ be the fixed mask value. Then, $K'^{\,i,r}$

satisfies $K'^{\,i,r} = K^i \oplus FK_{i,r}$.

**Algorithm 2** *ByteSub_FM*

*ByteSub_FM*$(X, r)$
1 $(x_{15}, x_{14}, \ldots, x_0) = X$;
2 $for(j = 0; j < 16; j++)\ x_j = S'_r[x_j]$;
3 $X = (x_{15}, x_{14}, \ldots, x_0)$;
4 *output* $X$;

**Algorithm 3** *SboxUpdate*

*SboxUpdate*$(S, r)$
1 $for(x = 0; x < 256; x++)\ S'[x] = S[x \oplus FIN_r] \oplus FOUT_r$;
2 *output* $S'$;

$$FK_{i,r} = \begin{cases} FIN_r, & i = 0 \\ MixColumn(ShiftRow(FOUT_r)) \oplus FIN_r, & i = 1, \ldots, Nr-1 \\ ShiftRow(FOUT_r), & i = Nr \end{cases}$$

# IV. Design

In this chapter, we first describe the more efficient method than previous fixed-value masking method. For cheap smart cards where SODPA resistance is not required, this method is sufficient. But, for high-end smart cards, SODPA should be considered. How to make the simple fixed-value masking method resistant to SODPA is described in the next section.

## 4.1 Simple Fixed-Value Masking

We propose simple fixed-value masking that requires less ROM and additional processing than previous fixed-value masking method. In the method, $q$ pairs of one byte masks and modified S-boxes are stored in ROM. Modified S-box is computed using the corresponding mask as in **Algorithm 6**. One pair consisting of a mask and a modified S-box is selected randomly at the start of encryption. All bytes of plaintext are masked by the same one byte mask. Masked plaintext is encrypted using the corresponding modified S-box. At the end of the algorithm, mask is applied once again and the result is the desired ciphertext. In Fig.4.1 and the following algorithms, $m_r$ $(r = 0, 1, \ldots, q-1)$ represents the selected mask. $r$ is the index for the chosen mask and modified S-box. $S$ is the original S-box and $S'$ is the modified S-box. $ByteSub\_FM$ is same with **Algorithm 2**.

**Algorithm 4** $SimpleFixedMasking_{AESEnc}$

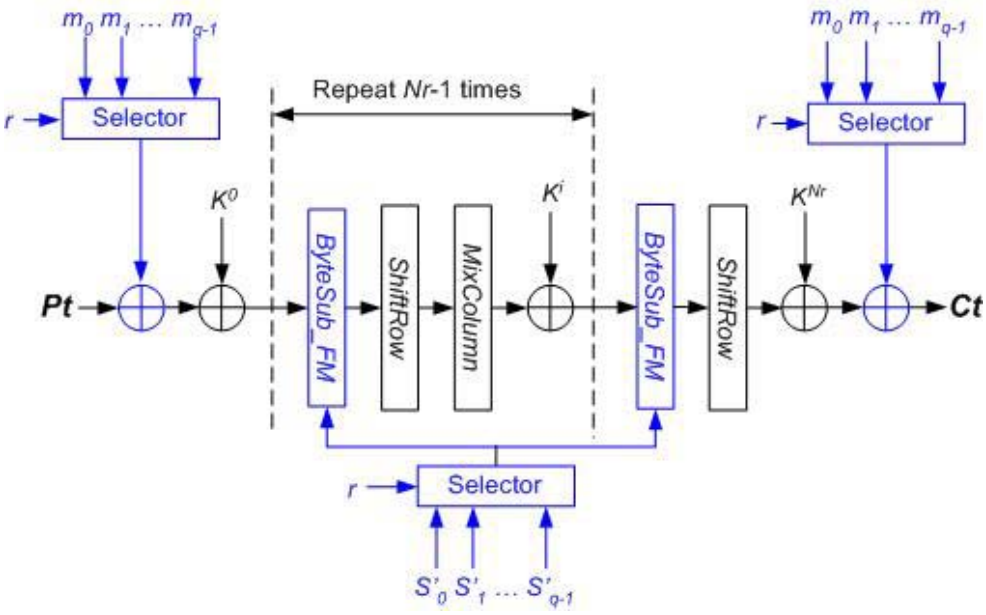


Figure 4.1: Simple fixed-value masking

$SimpleFixedMasking_{AESEnc}(Pt)$

1 $r = GenerateRandomNumber();\ /*\ choose\ r = 0, \ldots, q-1\ */$
2 $T' = Pt;$
3 $T' = ApplyMask(T', r);$
4 $T' = T' \oplus K^0;$
5 $for(i = 1; i < Nr; i++)\{$
6 $\quad T' = ByteSub\_FM(T', r);$
7 $\quad T' = ShiftRow(T');$
8 $\quad T' = MixColumn(T');$
9 $\quad T' = T' \oplus K^i;$
10 $\}$
11 $T' = ByteSub\_FM(T', r);$
12 $T' = ShiftRow(T');$
13 $T' = T' \oplus K^{Nr};$
14 $T = ApplyMask(T', r);$

15 *output* $T$;

**Algorithm 5** $ApplyMask$

$ApplyMask(T', r)$
1 $(t'_{15}, t'_{14}, \ldots, t'_0) = T'$;
2 $for(j = 0; j < 16; j++)\ t'_j = t'_j \oplus m_r$;
3 $T' = (t'_{15}, t'_{14}, \ldots, t'_0)$;
4 *output* $T'$;

**Algorithm 6** $SboxUpdate2$

$SboxUpdate2(S, r)$
1 $for(x = 0; x < 256; x++)\ S'[x] = S[x \oplus m_r] \oplus m_r$;
2 *output* $S'$;

### 4.1.1 Mask across Transformation

The reason why we can get the desired ciphertext at the end by applying the same mask used in the beginning is that the mask is maintained across transformations. Mask is not changed because modified S-box is used and the same mask is applied to all bytes of the State. We will show that the mask is maintained for each transformation used in AES.

#### ByteSub

During *ByteSub* transformation, the mask is maintained because the modified S-box is generated so that the mask does not change using **Algorithm 6**.

**ShiftRow**

$ShiftRow$ transformation does not change the mask because all the bytes in the State have the same mask and only the position is changed by $ShiftRow$ transformation.

**MixColumn**

If we consider $j$-th column of the State and denote each byte of the column by $a_{i,j}(i = 0, 1, 2, 3)$, $MixColumn$ transformation can be represented by the following equation as explained before.

$$\begin{bmatrix} b_{0,j} \\ b_{1,j} \\ b_{2,j} \\ b_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_{0,j} \\ a_{1,j} \\ a_{2,j} \\ a_{3,j} \end{bmatrix}$$

If each byte of the column is masked with $m_r$, the following equation can be derived.

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_{0,j} \oplus m_r \\ a_{1,j} \oplus m_r \\ a_{2,j} \oplus m_r \\ a_{3,j} \oplus m_r \end{bmatrix}$$

$$= \begin{bmatrix} b_{0,j} \\ b_{1,j} \\ b_{2,j} \\ b_{3,j} \end{bmatrix} \oplus m_r \begin{bmatrix} 02 \oplus 03 \oplus 01 \oplus 01 \\ 01 \oplus 02 \oplus 03 \oplus 01 \\ 01 \oplus 01 \oplus 02 \oplus 03 \\ 03 \oplus 01 \oplus 01 \oplus 02 \end{bmatrix} = \begin{bmatrix} b_{0,j} \oplus m_r \\ b_{1,j} \oplus m_r \\ b_{2,j} \oplus m_r \\ b_{3,j} \oplus m_r \end{bmatrix}$$

From the above equations, we can see that the mask before $MixColumn$ transformation is maintained after the transformation.

**AddRoundKey**

The mask is maintained through $AddRoundKey$ transformation because $(a_{i,j} \oplus m_r) \oplus k^l_{i,j} = (a_{i,j} \oplus k^l_{i,j}) \oplus m_r$ $(r = 0, 1, \ldots, q-1,\ i = 0, 1, 2, 3,\ j = 0, 1, 2, 3)$. $a_{i,j}$ is a byte of the State at $i$-th row and $j$-th column. And $m_r$ is the selected mask. $k^l_{i,j}$ is a byte of $l$-th round key at $i$-th row and $j$-th column.

### 4.1.2 Consideration for Implementation

If the masks and the modified S-boxes are stored in ROM, the values are known to the developer and are same across smart cards that have been produced with same mask. Maybe some attacks could be tried using this fact. If the masks are stored in EEPROM and the masks and the modified S-boxes are generated secretly and different from card to card, cards will be more secure. And the random number for selecting a mask can be acquired from hardware random number generator of smart cards. Because smart cards are used for security, even the cheap smart cards like S3C8982 of Samsung Electronics have hardware random number generator in them.

## 4.2 Securing against SODPA

Masking method is known to be subject to SODPA [8]. In this section, we propose how to make the masking method secure against SODPA.

### 4.2.1 SODPA

While masking method can be a countermeasure for DPA, it is vulnerable to SODPA. Let's consider the implementation of masking method as shown in **Algorithm 7**.

**Algorithm 7** *MaskedFunction*

*MaskedFunction*($T$)

1 $m = GenerateRandomNumber()$;
2 $maskedT = T \oplus m$;
3 $T' = maskedT \oplus SecretKey$;
4 ...
5 *other operations*
6 ...
7 *output* $T'$;

It is known that if there is a linear relationship between the instantaneous power consumption and the Hamming weight of the data being processed, **Algorithm 8** is sound [8] and it was verified with experiment. In **Algorithm 8**, $i$ is the position of one bit in SecretKey in **Algorithm 7**.

**Algorithm 8** *SecondOrderDPA*

*SecondOrderDPA*($i$)

1 $for(b = 0; b <= 1; b++)\{$
2 *Calculate average statistics* $\overline{S_b} = |P_1 - P_3|$ *by repeating*$\{$
3 *Set the* $i-th$ *bit of* $T$ *input to* $b$.
4 *Make remaining bits of* $T$ *random.*
5 *Collect algorithm's instantaneous power consumption at lines* 1 *and* 3. *Call these values* $P_1$ *and* $P_3$ *respectively.*
6 $\}$
7 $\}$
8 *Calculate the DPA bias statistic* $B = \overline{S_0} - \overline{S_1}$

9 $If\ B > 0\ then\ the\ i-th\ key\ bit\ is\ one, otherwise,\ it\ is\ zero$

### 4.2.2 Modification of Simple Fixed-Value Masking

We can avoid SODPA by making it difficult for an attacker to decide the point where mask is loaded. For this, we can load more than two masks into memory that can be used for operand of xor instruction. And then we can use one mask among them randomly. SODPA resistant simple fixed-value masking method is shown in **Algorithm 9**.

**Algorithm 9** $SODPAResistantSFVM_{AESEnc}$

$SODPAResistantSFVM_{AESEnc}(Pt)$

1 $r[0] = GenerateRandomNumber();\ /*\ r[0] = 0, \ldots, q-1\ */$
2 $r[1] = GenerateRandomNumber();\ /*\ r[1] = 0, \ldots, q-1\ */$
3 $lm[0] = m_{r[0]};$
4 $lm[1] = m_{r[1]};$
5 $s = GenerateRandomNumber(); /*\ choose\ s = 0\ or\ 1\ */$
6 $T' = Pt;$
7 $T' = ApplyMask2(T', s);$
8 $T' = T' \oplus K^0;$
9 $for(i = 1; i < Nr; i++)\{$
10 $\quad T' = ByteSub\_FM(T', r[s]);$
11 $\quad T' = ShiftRow(T');$
12 $\quad T' = MixColumn(T');$
13 $\quad T' = T' \oplus K^i;$
14 $\}$
15 $T' = ByteSub\_FM(T', r[l]);$
16 $T' = ShiftRow(T');$
17 $T' = T' \oplus K^{Nr};$

18 $T = ApplyMask2(T', s)$;
19 *output* $T$;

**Algorithm 10** $ApplyMask2$

$ApplyMask2(T', s)$
1 $(t'_{15}, t'_{14}, \ldots, t'_0) = T'$;
2 $for(j = 0; j < 16; j++)\ t'_j = t'_j \oplus lm[s]$;
3 $T' = (t'_{15}, t'_{14}, \ldots, t'_0)$;
4 *output* $T'$;

Because the attacker cannot know which mask in the memory is used, he cannot decide the point for SODPA.

### 4.2.3 Consideration for Implementation

An 8-bit microcontroller like ST72101 has xor instruction. The source operand can be accumulator and memory address. The destination operand can be accumulator. In implementing $ApplyMask2$, care should be taken so that mask value is not moved from the place where it is stored by line 3 or 4 in **Algorithm 9**. If it is moved to other memory, the moment can be a target for SODPA. If the mask is not moved and used in the place using the address of the mask, an attacker will not be able to find a point to mount SODPA.

### 4.2.4 Less Efficient but Safer Method

In the method of previous sections, it may be possible for an attacker to attack the point where mask is moved into ALU of microcontroller. Each instruction has different degree of vulnerability to DPA and it seems that the xor instruction is relatively less vulnerable to DPA. But

we can use the following algorithm for more security.

**Algorithm 11** $SODPAResistantSFVM2_{AESEnc}$

$SODPAResistantSFVM2_{AESEnc}(Pt)$

1 $r[0] = GenerateRandomNumber();\ /*\ r[0] = 0, \ldots, q-1\ */$

2 $r[1] = GenerateRandomNumber();\ /*\ r[1] = 0, \ldots, q-1\ */$

3 $lm[0] = m_{r[0]};$

4 $lm[1] = m_{r[1]};$

5 $s = GenerateRandomNumber(); /*\ choose\ s = 0\ or\ 1\ */$

6 $Pt[0] = ApplyMask2(Pt, 0);$

7 $Pt[1] = ApplyMask2(Pt, 1);$

8 $T' = Pt[s];$

9 $T' = T' \oplus K^0;$

10 $for(i = 1; i < Nr; i++)\{$

11 $\quad T' = ByteSub\_FM(T', r[s]);$

12 $\quad T' = ShiftRow(T');$

13 $\quad T' = MixColumn(T');$

14 $\quad T' = T' \oplus K^i;$

15 $\}$

16 $T' = ByteSub\_FM(T', r[l]);$

17 $T' = ShiftRow(T');$

18 $T' = T' \oplus K^{Nr};$

19 $Ct[0] = ApplyMask2(T', 0);$

20 $Ct[1] = ApplyMask2(T', 1);$

21 $output\ Ct[s];$

Because two masked plaintext is prepared and only the selected one is processed, an attacker will not be able to decide the mask loading time.

# V. Analysis

## 5.1 Security Analysis

### 5.1.1 Resistance against DPA

To apply DPA, there should be an intermediate value that is function of plaintext (or ciphertext) and part of secret key. If this condition is met, we can find out the secret key part by part. But in simple fixed-value masking method, all the bytes in the State are masked with randomly selected mask and an attacker cannot know which value is used for masking. Therefore, the attacker cannot know the bytes of the State from plaintext and part of key. Because the mask is maintained until the final round key addition finishes, the attacker cannot guess the bytes of the State before final round key addition from the ciphertext.

If we choose the masks so that any bit of mask can be 0 with probability of 1/2, simple fixed-value masking method will be secure against probabilistic DPA proposed in [4]. The proof will be similar to that of [4]. We can summarize the proof for probabilistic DPA on S-box and on round key addition as follows:

#### Proof of Resistance against DPA on S-box

If we denote the probability of $d$-th bit of $x$ being 0 as $\beta_d(x)$, the proof is as follows.

The relation between the output of modified S-box $S'$ and the original S-box $S$ of first round is like this.

$$S'[a_{i,j} \oplus m_r \oplus k^0_{i,j}] = S[a_{i,j} \oplus m_r \oplus k^0_{i,j} \oplus m_r] \oplus m_r = S[a_{i,j} \oplus k^0_{i,j}] \oplus m_r$$

The attacker guesses that the key is $k^0_{i,j}$ and calculates the differential power consumption curve $\Delta_d$ with the selected bit from $f = S[a_{i,j} \oplus k^0_{i,j}]$. $\epsilon_{S'}$, the magnitude of the spike in differential power consumption for one byte output of $S'$ can be represented as follows:

$$\epsilon_{S'} = \frac{2}{N}\left\{ \sum_{V_c \in \delta 1_d(S[a_{i,j} \oplus k^0_{i,j}])} V_c(t_{loadS'}) - \sum_{V_c \in \delta 0_d(S[a_{i,j} \oplus k^0_{i,j}])} V_c(t_{loadS'}) \right\}$$

Because $d$-th bit of $S[a_{i,j} \oplus k^0_{i,j}]$ equals to $d$-th bit of $S[a_{i,j} \oplus k^0_{i,j}] \oplus m_r$ with probability $\beta_d(m_r)$, we can get the following equation.

$$\begin{aligned}
\epsilon_{S'} &= \frac{2}{N}\left\{ \beta_d(m_r) \sum_{V_c \in \delta 1_d(S[a_{i,j} \oplus k^0_{i,j}] \oplus m_r)} V_c(t_{loadS'}) \right. \\
&\quad + (1 - \beta_d(m_r)) \sum_{V_c \in \delta 0_d(S[a_{i,j} \oplus k^0_{i,j}] \oplus m_r)} V_c(t_{loadS'}) \\
&\quad - \beta_d(m_r) \sum_{V_c \in \delta 0_d(S[a_{i,j} \oplus k^0_{i,j}] \oplus m_r)} V_c(t_{loadS'}) \\
&\quad \left. - (1 - \beta_d(m_r)) \sum_{V_c \in \delta 1_d(S[a_{i,j} \oplus k^0_{i,j}] \oplus m_r)} V_c(t_{loadS'}) \right\} \\
&= \frac{2(2\beta_d(m_r) - 1)}{N}\left\{ \sum_{V_c \in \delta 1_d(S'[a_{i,j} \oplus m_r \oplus k^0_{i,j}])} V_c(t_{loadS'}) \right. \\
&\quad \left. - \sum_{V_c \in \delta 0_d(S'[a_{i,j} \oplus m_r \oplus k^0_{i,j}])} V_c(t_{loadS'}) \right\}
\end{aligned}$$

From the above equation, we can see that the magnitude of spike is zero if $\beta_d(m_r)$ is 1/2, which means that any bit of mask should be 0 with probability 1/2.

**Proof of Resistance against DPA on round key addition**

The attacker calculates the differential power consumption curve $\Delta_d$ with the selected bit from $f = a_{i,j}$. We assume that the sequence of operation for applying round key is as follows:

`load` $a_{i,j}$
`load` $k^0_{i,j}$
`xor` $t_{i,j}$, $a_{i,j}$, $k^0_{i,j}$
`store` $t_{i,j}$
`load` $t_{i,j}$

The magnitude of the spike for load $t_{i,j}$ can be as follows:

$$\epsilon_{loadt_{i,j}} = \frac{2}{N}\left\{\sum_{V_c \in \delta 1_d(a_{i,j})} V_c(t_{loadt_{i,j}}) - \sum_{V_c \in \delta 0_d(a_{i,j})} V_c(t_{loadt_{i,j}})\right\}$$

Considering that $d$-th bit of $a_{i,j}$ equals to $d$-th bit of $a_{i,j} \oplus m_r$ with probability $\beta_d(m_r)$, the magnitude of spike is as follows:

$$\begin{aligned}\epsilon_{loadt_{i,j}} = \frac{2}{N}\Bigg\{ & \beta_d(m_r) \sum_{V_c \in \delta 1_d(a_{i,j} \oplus m_r)} V_c(t_{loadt_{i,j}}) \\ & + (1-\beta_d(m_r)) \sum_{V_c \in \delta 0_d(a_{i,j} \oplus m_r)} V_c(t_{loadt_{i,j}}) \\ & - \beta_d(m_r) \sum_{V_c \in \delta 0_d(a_{i,j} \oplus m_r)} V_c(t_{loadt_{i,j}}) \\ & - (1-\beta_d(m_r)) \sum_{V_c \in \delta 1_d(a_{i,j} \oplus m_r)} V_c(t_{loadt_{i,j}}) \Bigg\}\end{aligned}$$

$$= \frac{2(2\beta_d(m_r)-1)}{N}\left\{\sum_{V_c\in\delta 1_d(a_{i,j}\oplus m_r)} V_c(t_{loadt_{i,j}}) - \sum_{V_c\in\delta 0_d(a_{i,j}\oplus m_r)} V_c(t_{loadt_{i,j}})\right\}$$

In the above equation, if the value of $\beta_d(m_r)$ is 1/2, which means that $d$-th bit of mask will be zero with probability 1/2, the magnitude of the spike is zero and the attacker cannot guess the key bit.

### 5.1.2 Resistance against SODPA

In the proposed method that is resistant to SODPA, the attacker cannot know the mask loading time. Therefore, the attack proposed by Messerges [8] cannot be applied because the attacker should know the power consumption by the mask.

It is said that if same mask is used for more than one round, mutual correlation can be used to mount SODPA [6]. To avoid this attack, mask can be selected for each round. At the point of mask change, the xor-ed value of the old mask and the new mask can be applied to each byte of the State.

### 5.1.3 Number of Masks

To apply the proposed method in real smart cards, the number of masks $q$ should be determined. If we use only one mask, the value of the mask should be 0xFF to satisfy the condition for resistance against probabilistic DPA. But Akkar *et al.* [12] showed that a variant of DPA called PODPA can be used to attack this situation. But to mount PODPA, an attacker should know the precise power consumption characteristic of the target device and this is not an easy task for a normal attacker.

Therefore, for low-end smart cards, the number of masks $q$ can be one. But for high-end smart cards, $q$ should be greater than one. And to satisfy the criteria for probabilistic DPA resistance, $q$ should be even. It seems that there is no known attack for $q = 2$ case. But for security margin, we prefer that $q$ is greater than or equal to 4.

## 5.2 Performance Analysis

For the DPA resistance, a mask and a modified S-box can be used for an entire encryption. In this case, additional processing and memory requirement is as follows. First, random number for selecting a mask should be generated. Then, the mask should be xor-ed with each byte of the State. Encryption can be done as usual except that the selected modified S-box should be used instead. At the end of the encryption, the mask is applied once again and the result is the required ciphertext. In this flow, the additional processing consists of random number generation, initial mask application and final mask application. The additional memory will be one or two bytes of register and additional processing is random number generation and 32 xor operations.

For SODPA resistance, more resource is required. For hiding the mask loading time, three random numbers should be generated and two masked plaintext should be prepared and two unmasked ciphertext should be calculated. For this, about 40 bytes of additional memory and 64 additional xor operations are needed. For changing the mask across rounds, new mask should be applied $Nr$ times and $Nr * (1 + 16) * 2$ xor operations are needed additionally to calculate the 2 xor-ed values of the old masks and the new masks and apply the calculated values to each bytes of States. The number 2 accounts for the two masked States.

# VI. Comparison

In this chapter, we compare the proposed method with existing fixed-value masking that is most adequate for real smart cards.

## 6.1 Viewpoint

The proposed method is different from fixed-value masking in the viewpoint of mask generation. In fixed-value masking, input mask and output mask for S-box are determined and from those masks, masks for round keys are derived. By following this approach, applied mask can be changed only at fixed points. That is at the place where round key is added or where *ByteSub* transformation is performed.

In the proposed method, mask for bytes of the State is generated regardless of other parts. And the applied mask can be changed at any points in the algorithm by applying the value generated by xor operation of the old mask and the new mask.

## 6.2 Security

In fixed-value masking method, S-box input mask $FIN$ is used in the interval from $AddRoundKey$ transformation to *ByteSub* transformation. S-box output mask $FOUT$ is used in the interval from *ByteSub* transformation to $AddRoundKey$ transformation. Two masks are used repeatedly across rounds. It seems that using two masks for one round has no advantage against DPA. Therefore, the proposed method without SODPA countermeasures is as secure as fixed-value masking method.

The proposed method has advantage against SODPA especially proposed by Messerges [8]. In the proposed method, it is difficult for an attacker to decide the mask loading time, and SODPA proposed by Messerges is impossible. SODPA using the mutual correlation across rounds [6] can be prevented more efficiently using the proposed method. To be able to select mask for each round, three bytes should be stored with the modified S-box in fixed-value masking. One byte is $FIN_r$. This is used for round key mask when the set of masks and modified S-box is selected for the first round. When the mask set is selected for intermediate rounds, $FIN_r$ is used for generating the round key mask that is the xor-ed value of $FIN_r$ and $MixColumn(ShiftRow(FOUT_r))$ in the previous round. Another byte is $ShiftRow(FOUT_r)$. This is used for round key mask when the set of masks and modified S-box is used for the final round. The third is $MixColumn(ShiftRow(FOUT_r))$. This is used for round key generation of intermediate rounds as explained in the first case.

But with the proposed method, only one byte mask $m_r$ is stored to be resistant to SODPA proposed at [6]. Mask can be changed at any point simply by applying the xor-ed value of the old mask and the new mask to every byte of the State. When a modified S-box is needed, the modified S-box that corresponds to the current mask can be selected from ROM or EEPROM and can be used.

## 6.3 Performance

The proposed method has advantage in memory and processing requirement. Previous fixed-value masking method requires three bytes to store one kind of mask. That is, one for initial round key, another for intermediate round keys, and the third for final round key. But the proposed method uses only one byte mask.

If we consider the case without SODPA resistance for comparison, the proposed method requires less processing because mask is applied only at two points. That is, before initial round key addition and after final round key addition. But the fixed-value masking method applies masks at ($Nr$ + 1) points. Once at the initial round key addition, and ($Nr$ - 1) times at intermediate round key addition and once at final round key addition. Moreover, if the key becomes larger than 128 bits, it requires more xor computation at one point. For AES whose secret key is 128 bits long, the number of xor operations for applying mask is like this. In fixed-value masking method, it is 11 points * 16 bytes = 176 operations. In the proposed method, it is 2 points * 16 bytes = 32 operations. We summarize the comparison in **Table 6.1**.

In some applications of smart cards like in transportation, the speed of processing is important. All transactions should be completed without incurring user's inconvenience. Because the processing power of smart cards are relatively low and cryptographic processing occupies much part of the transaction, simple implementation of cryptographic algorithm is required. And recently, some 32 bit smart card chips like SLE88CX720P of Infineon technologies are beginning to be used. According to the AES proposal of Rijndael [15], if we implement AES-128 in 32 bit processors with full optimization for speed, AES-128 can be implemented with four table lookups and four xor operations for each column of the State in a round. In such an implementation of AES-128, sixteen table lookups and sixteen xor operations are needed for a round. For entire encryption, four word size xor operations are needed for the initial round key addition and 320 operations for the other parts. And the total number of operations will be approximately 324. If we use fixed-value masking, four additional word size xor operations are needed for each round key. For the entire encryption, 4 * 11 = 44 xor operations are needed additionally. This increases the overall number of operation

Table 6.1: Comparison of fixed-value masking and our scheme for AES-128

| | Fixed-value masking | Our scheme |
|---|---|---|
| ROM space for mask | 3 Bytes/mask | 1 Byte/mask |
| Number of xor for masking | 176 | 32 |

by 13.6%. If we use the proposed method, 4 * 2 = 8 additional word size xor operations are needed for masking the columns of the State. The increase in the number of operations is 2.5%. From this, we can see that the increase in processing for applying the countermeasure is much less with the proposed method.

# VII. Conclusion

In this thesis, we have studied the design and analysis of software countermeasure for securing AES against DPA. We have reviewed AES, DPA, and previous masking methods. And then we have suggested an improved masking method.

Fixed-value masking method seems to be the most practical software countermeasure for securing AES against DPA. The method is especially adequate for cheap smart cards that have little RAM and low processing power. But the method is vulnerable to SODPA and it can be more simplified maintaining security. In this thesis, we propose a SODPA resistant simple fixed-value masking method. In the proposed method, we reduced the required memory for storing mask by 66% and the xor operation for applying mask by 82%. By making it difficult to determine the point for SODPA, we achieved SODPA resistance. For more resistance against SODPA, we can select the mask for each round. The additional RAM usage and processing for DPA resistance is relatively small.

As the further works, it is necessary to implement proposed method in real smart cards and assess its effectiveness. And the possibility of other SODPA other than considered in this thesis should be checked and study of DPA higher than second order is needed.

# 차분전력공격에 대한 AES 안전성 강화 방법 연구

장화선

가까운 시일 내에 대부분의 신용카드가 스마트카드로 바뀔 전망이며 교통카드, 전자화폐, 전자신분증 등 스마트카드의 이용은 증가할 것으로 예상된다. 스마트카드의 주요한 장점은 비밀키 등을 이용한 연산을 내부적으로 수행한 후, 그 결과만을 외부로 전송하여 내부 데이터를 안전하게 보호할 수 있다는 것이다.

하지만, 최근 부채널 공격이 알려지면서 스마트카드의 안전성이 위협받고 있다. 부채널 공격은 입력 메시지와 비밀키로부터 출력 메시지를 만들면서 발생하는 누설 정보를 이용하여 비밀키를 알아내는 기술이다. 누설 정보로는 전력 소모나 전자기파 등을 이용하기도 하고 연산 시간이나 연산 중 발생하는 오류를 이용하기도 한다. 차분전력공격은 전력 소모를 이용하는 부채널 공격의 일종으로 비교적 저가의 장비로 알고리즘 내부 구현을 모르면서도 적용이 가능하여 스마트카드 같은 장치에 대해서 실질적인 위협이 되고 있다. 차분전력공격에 대한 대응방안은 하드웨어에 의한 방법과 소프트웨어에 의한 방법으로 나눌 수 있다. 최근 생산되는 스마트카드는 칩 자체에 차분전력공격을 막기 위한 하드웨어 대응방안이 구현되어 있는 경우가 많지만, 차분전력공격을 효과적으로 방지하기 위해서는 소프트웨어에 의한 대응방안도 구현해야 한다는 것이 일반적인 의견이다.

AES는 미국 국립표준기술원(NIST)이 DES를 대신할 새로운 표준 블록 암호 알고리즘으로 2000년에 선정한 알고리즘으로 앞으로 사용이 점차 증가할 전망이다. AES를 차분전력공격으로부터 보호하기 위한 소프트웨어 대응방안으로 마스크 방법이 제안되었지만 기존 마스크 방법은 이차 차분전력공격에 대해 유효성이 없고 메모리 사용량과

연산량 면에서도 효율성을 높일 여지가 있다.

본 논문에서는 이차 차분전력공격에 견딜 수 있고 기존 방법보다 효율적인 단순 마스크 방법을 제안한다. 제안한 방법은 마스크 자체 저장을 위한 메모리 사용량에서 기존 방법의 33%이며 마스크 적용을 위한 xor 연산은 128비트 키를 사용하는 AES 기준으로 기존 방법의 18%이다. 마스크에 의한 메모리 사용량을 줄인 것은 실제 구현 시 알고리즘 전체 크기에 비하면 효과가 크지 않다. 하지만 xor 연산을 줄인 것은 속도에 최적화된 32비트 스마트카드에서는 전체 수행 속도를 10% 정도 향상시키는 효과가 있다. 이차 차분전력공격을 방지하기 위해서는 공격자가 실제 사용된 마스크의 이동 시점을 알 수 없도록 하는 방법과 라운드별로 마스크를 바꾸는 방법을 제안한다. 분석 부분에서는 생성된 마스크가 가져야 할 요건과 마스크의 수, 대응방안 구현에 따른 추가 연산량과 메모리 사용량을 제시하고 안전성을 검증한다.

# References


1. P. Kocher, J. Jaffe, and B. Jun, Differential Power Analysis, *Advances in Cryptology - Crypto 1999*, LNCS 1666, pp.388-397, Springer-Verlag, 1999.

2. T. Messerges, Securing the AES Finalists Against Power Analysis Attacks, *Fast Software Encryption Workshop - FSE 2000*, LNCS 1978, pp.150-164, Springer-Verlag, 2001.

3. C. Clavier, J. Coron, and N. Dabbous, Differential Power Analysis in the Presence of Hardware Countermeasures, *Workshop on Cryptographic Hardware and Embedded Systems - CHES 2000*, LNCS 1965, pp.252-263, Springer-Verlag, 2000.

4. K. Itoh, M. Takenaka, and N. Torii, DPA Countermeasure Based on the Masking Method, *International Conference on Information, Communications and Signal Processing - ICICS 2001*, LNCS 2288, pp.440-456, Springer-Verlag, 2002.

5. M. Akkar and C. Giraud, An Implementation of DES and AES Secure against Some Attacks, *Workshop on Cryptographic Hardware and Embedded Systems - CHES 2001*, LNCS 2162, pp.309-318, Springer-Verlag, 2001.

6. J. Golic and C. Tymen, Multiplicative Masking and Power Analysis of AES, *Workshop on Cryptographic Hardware and Embedded Systems - CHES 2002*, LNCS 2523, pp.198-212, Springer-Verlag, 2003.

7. E. Trichina, D. Seta, and L. Germani, Simplified Adaptive Multiplicative Masking for AES, *Workshop on Cryptographic Hardware and Embedded Systems - CHES 2002*, LNCS 2523, pp.187-197, Springer-Verlag, 2003.

8. T. Messerges, Using Second-Order Power Analysis to Attack DPA Resistant Software, *Workshop on Cryptographic Hardware and Embedded Systems - CHES 2000*, LNCS 1965, pp.238-251, Springer-Verlag, 2000.

9. S. Yen, Amplified Differential Power Cryptanalysis on Rijndael Implementations with Exponentially Fewer Power Traces, *Information Security and Privacy Australasian Conference - ACISP 2003*, LNCS 2727, pp.106-117, Springer-Verlag, 2003.

10. Y. Ishai, A. Sahai, and D. Wagner, Private Circuits: Securing Hardware against Probing Attacks, *Advances in Cryptology - Crypto 2003*, LNCS 2729, pp.463-481, Springer-Verlag, 2003.

11. NIST, Federal Information Processing Standards Publication 197 - Specification for the Advanced Encryption Standard (AES), *http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf*, 2001.

12. M. Akkar, R. Bevan, P. Dischamp, and D. Moyart, Power Analysis, What Is Now Possible..., *Advances in Cryptology - ASIACRYPT 2000*, LNCS 1976, pp.489-502, Springer-Verlag, 2000.

13. S. Chari, C. Jutla, J. Rai, and P. Rohatgi, A Cautionary Note Regarding Evaluation of AES Candidates on Smart-Cards, *Second AES Candidate Conference (AES2)*, 1999.

14. R. Bevan and E. Knudsen, Ways to Enhance Differential Power Analysis, *Information Security and Cryptology - ICISC 2002*, LNCS 2587, pp.327-342, 2003.

15. J. Daemen and V. Rijmen, AES Proposal: Rijndael, *http://csrc.nist.gov/CryptoToolkit/aes/rijndael/Rijndael-ammended.pdf*, 1999.

16. J. Coron and L. Goubin, On Boolean and Arithmetic Masking against Differential Power Analysis, *Workshop on Cryptographic Hardware and Embedded Systems - CHES 2000*, LNCS 1965, pp.231-237, Springer-Verlag, 2000.

17. L. Goubin, A Sound Method for Switching between Boolean and Arithmetic Masking, *Workshop on Cryptographic Hardware and Embedded Systems - CHES 2001*, LNCS 2162, pp.3-15, Springer-Verlag, 2001.

18. J. Coron, A. Tchulkine, A New Algorithm for Switching from Arithmetic to Boolean Masking, *Workshop on Cryptographic Hardware and Embedded Systems - CHES 2003*, LNCS 2779, pp.89-97, Springer-Verlag, 2003.

19. T. Messerges, E. Dabbish, and R. Sloan, Investigations of Power Analysis Attacks on Smartcards, *USENIX Workshop on Smartcard Technology*, pp.151-162, 1999.

20. P. Kocher, Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems, *Advances in Cryptology - Crypto 1996*, LNCS 1109, pp.104-113, Springer-Verlag, 1996.

21. W. Schindler, F. Koeune, and J. Quisquater, Improving Divide and Conquer Attacks against Cryptosystems by Better Error Detection / Correction Strategies, *Cryptography and Coding 2001*, LNCS 2260, pp.245-267, Springer-Verlag, 2001.

22. A. Shamir, Protecting Smart Cards from Passive Power Analysis with Detached Power Supplies, *Workshop on Cryptographic Hard-*

*ware and Embedded Systems - CHES 2000*, LNCS 1965, pp.71-77, Springer-Verlag, 2000.

23. S. Mangard, A Simple Power-Analysis (SPA) Attack on Implementations of the AES Key Expansion, *Information Security and Cryptology - ICISC 2002*, LNCS 2587, pp.343-358, 2003.

24. S. Chari, C. Jutla, J. Rao, and P. Rohatgi, Towards Sound Approaches to Counteract Power-Analysis Attacks, *Advances in Cryptology - Crypto 1999*, LNCS 1666, pp.398-412, Springer-Verlag, 1999.

25. W. Schindler, A Combined Timing and Power Attack, *International Workshop on Practice and Theory in Public Key Cryptography - PKC 2002*, LNCS 2274, pp.263-279, 2002.

26. M. Willich, A Technique with an Information-Theoretic Basis for Protecting Secret Data from Differential Power Attacks, *Cryptography and Coding 2001*, LNCS 2260, pp.44-62, 2001.

27. S. Yen, S. Kim, S. Lim, and S. Moon, A Countermeasure against One Physical Cryptanalysis May Benefit Another Attack, *International Conference on Information, Communications and Signal Processing - ICICS 2001*, LNCS 2288, pp.414-427, Springer-Verlag, 2002.

28. R. Sommer, Smartly Analyzing the Simplicity and the Power of Simple Power Analysis on Smartcards, *Workshop on Cryptographic Hardware and Embedded Systems - CHES 2000*, LNCS 1965, pp.78-92, Springer-Verlag, 2000.

29. D. May, H. Muller, and N. Smart, Random Register Renaming to Foil DPA, *Workshop on Cryptographic Hardware and*

*Embedded Systems - CHES 2001*, LNCS 2162, pp.28-38, Springer-Verlag, 2001.

30. L. Goubin and J. Patarin, DES and Differential Power Analysis The "Duplication" Method, *Workshop on Cryptographic Hardware and Embedded Systems - CHES 1999*, LNCS 1717, pp.158-172, Springer-Verlag, 1999.

31. V. Klíma and T. Rosa, Further Results and Considerations on Side Channel Attacks on RSA, *Workshop on Cryptographic Hardware and Embedded Systems - CHES 2002*, LNCS 2523, pp.244-259, Springer-Verlag, 2003.

32. C. Walter and S. Thompson, Distinguishing Exponent Digits by Observing Modular Substractions, *Cryptographer's Track at RSA Conference - CT-RSA 2001*, LNCS 2020, pp.192-207, 2001.

33. R. Novak, SPA-Based Adaptive Chosen-Ciphertext Attack on RSA Implementation, *International Workshop on Practice and Theory in Public Key Cryptography - PKC 2002*, LNCS 2274, pp.252-262, 2002.

34. J. Dhem, F. Koeune, P. Leroux, P. Mestré, J. Quisquater, and J. Willems, A Practical Implementation of the Timing Attack, *UCL Crypto Group Technical Report http://users.belgacom.net/dhem/papers/CG1998_1.pdf*, 1998.

35. P. Fouque, G. Martinet, and G. Poupard, Attacking Unbalanced RSA-CRT Using SPA, *Workshop on Cryptographic Hardware and Embedded Systems - CHES 2003*, LNCS 2779, pp.254-268, Springer-Verlag, 2003.

36. W. Schindler, A Timing Attack against RSA with the Chinese Remainder Theorem, *Workshop on Cryptographic Hardware*

*and Embedded Systems - CHES 2000*, LNCS 1965, pp.109-124, Springer-Verlag, 2000.

37. K. Gandolfi, C. Mourtel, and F. Olivier, Electromagnetic Analysis: Concrete Results, *Workshop on Cryptographic Hardware and Embedded Systems - CHES 2001*, LNCS 2162, pp.251-261, Springer-Verlag, 2001.

38. D. Page, Theoretical Use of Cache Memory as a Cryptanalytic Side-Channel, *http://eprint.iacr.org/2002/169*, 2002.

39. C. Giraud, DFA on AES, *http://eprint.iacr.org/2003/008*, 2003.

40. S. Skorobogatov and R. Anderson, Optical Fault Induction Attacks, *Workshop on Cryptographic Hardware and Embedded Systems - CHES 2002*, LNCS 2523, pp.2-12, Springer-Verlag, 2003.

41. R. Anderson and M. Kuhn, Low Cost Attacks on Tamper Resistant Devices, *Security Protocols 1997*, LNCS 1361, pp.125-136, Springer-Verlag, 1997.

42. E. Biham and A. Shamir, Differential Fault Analysis, *Advances in Cryptology - Crypto 1997*, LNCS 1294, pp.513-525, Springer-Verlag, 1997.

43. R. Anderson and M. Kuhn, Tamper Resistance - a Cautionary Note, *The Second Workshop on Electronic Commerce*, pp.1-11, 1996.

44. D. Boneh, R. DeMillo, and R. Lipton, On the Importance of Checking Cryptographic Protocols for Faults, *Advances in Cryptology - EUROCRYPT 1997*, LNCS 1233, pp.37-51, Springer-Verlag, 1997.

# Acknowledgement

First, I would like to express my sincere gratitude to Prof. Kwangjo Kim, my academic advisor, for his constant direction and support. He always has shown his consistent affection and encouragement for me to carry out my research and life in ICU. Special thanks also goes to Prof. Jaechoon Cha and Prof. Daeyoung Kim for their generosity and agreeing to serve as committee members of my thesis.

In particular, I would like to thank to people in my company, Samsung SDS for giving me a good chance to do research.

I also would like to thank to all members of cryptology and information security laboratory: Kyusuk Han, Sangwon Lee, Zeen Kim, Byeonggon Kim, Songwon Lee, Chuljoon Choi, Sungjoon Min, Jaehyrk Park, Jungman Kim, Sugil Choi, Jungkyu Yang, Seokkyu Kang, Jeongmi Choi, Vo Duc Liem, Yan Xie, Xiaofeng Chen, Ping Wang, Jiqiang Lv, Ren Kui and Divyan, for giving me lots of interests and good advices during the course of my study.

In addition, I appreciate to the graduates, Weonkeun Huh, Kyubeom Hwang, Gookwhan An, Byoungcheon Lee, Manho Lee, Jinho Kim, Myungsun Kim, Jongseung Kim, Wooseok Ham, Hyunrok Lee, and Hyungki Choi for their everlasting guidance in life and study in ICU and I want to present my sincere gratitude to previous C Lab. members: Sangbae Park, Jungyun Lee, YunKyoung Jeong, and Junbaek Ki.

Most of all, I should mention my wife, Yunmi, Yujung, father, mother, father-in-law, mother-in-law, brothers, and brother-in-laws for their endless concerns and devotional affection. I cannot forget their trust and encouragement on me. I hope God bless my family and to be happy.

# Curriculum Vitae

Name : Hwasun Chang

Date of Birth : Nov. 06. 1970

Sex : Male

Nationality : Korean

## Education

| | |
|---|---|
| 1989.3–1993.2 | Electrical Engineering<br>Yonsei University (B.E.) |
| 1993.3–1995.2 | Electrical Engineering<br>Yonsei University (M.E.) |
| 2002.2–2004.2 | Cryptology and Information Security, Engineering<br>Information and Communications University (M.S.) |

## Career

| | |
|---|---|
| 2003.6–2003.8 | Graduate Teaching Assistant<br>ICE1210 Algorithm Design and Analysis<br>School of Engineering, ICU |

| | |
|---|---|
| 2003.1–2003.12 | Graduate Research Assistant<br>Research on Link Layer Security Algorithm Development and Standardization<br>Electronics and Telecommunications Research Institute(ETRI) |
| 2002.1–2002.12 | Graduate Research Assistant<br>Middleware(8)<br>Electronics and Telecommunications Research Institute(ETRI) |
| 2002.2–2003.2 | Graduate Research Assistant<br>Cultivation of Top Level IT Security Manpower<br>The Ministry of Information and Communications(MIC) |
| 2003.3– | Graduate Research Assistant<br>Support for Running the International Research Center for Information Security<br>The Ministry of Information and Communications(MIC) |
| 1995.2–1998.6 | Samsung Electronics Co., Ltd. |
| 1998.7– | Samsung SDS Co., Ltd. |

## Academic Experience

| | |
|---|---|
| 2002.4– | KIISC student member |

## Publications

(1) 2003.10 Hwasun Chang and Kwangjo Kim, Securing AES against Second-Order DPA by Simple Fixed-Value Masking, *Computer Security Symposium 2003*, pp.145-150, Kitakyushu, Japan (학생논문상)

(2) 2003.12 장화선, 박상배, 김광조, SEED의 차분전력공격에 대한 강화 방안, 2003년도 한국정보보호학회 동계학술대회, pp.274-280, 한양대학교, 한국

(3) 2003.8 장화선, 김광조, AES에 대한 DPA 대응방안 개선, 2003년도 한국정보보호학회 충청지부 학술대회, pp.3-10, 충북대학교, 한국

(4) 2003.7 장화선, 김광조, AES에 대한 오류기반 공격, 2003년도 한국정보보호학회 하계학술대회, pp.138-141, 배재대학교, 한국