

Efficient Masking Methods Appropriate for the Block Ciphers ARIA and AES

HeeSeok Kim, Tae Hyun Kim, Dong-Guk Han, and Seokhie Hong

In this paper, we propose efficient masking methods for ARIA and AES. In general, a masked S-box (MS) block can be constructed in different ways depending on the implementation platform, such as hardware and software. However, the other components of ARIA and AES have less impact on the implementation cost. We first propose an efficient masking structure by minimizing the number of mask corrections under the assumption that we have an MS block. Second, to make a secure and efficient MS block for ARIA and AES, we propose novel methods to solve the table size problem for the MS block in a software implementation and to reduce the cost of a masked inversion which is the main part of the MS block in the hardware implementation.

Keywords: Side-channel attacks, masking method, composite field, ARIA, AES.

I. Introduction

Since Kocher introduced a differential power analysis (DPA) in 1998 [1], the security of block ciphers AES and ARIA have received considerable attention, and it is now obvious that unprotected implementations of these ciphers in embedded processors can be broken by DPA. This weakness has been exposed, for example, in [2] and [3], which describe DPA attacks on ARIA and AES software implementations, and in [4] and [5], which discuss DPA attacks on hardware implementations.

During the past few years, much of the research on DPA attacks has focused on finding secure countermeasures. Among these countermeasures, a masking method based on algorithmic techniques is known to be inexpensive and secure against a first-order DPA (FODPA). In a masking method, all intermediate values are concealed by a random value, which is called a mask. This makes it impossible to apply FODPA. While masking methods can prevent FODPA, these are known to still have a weakness for the high-order DPA (HODPA) [6]. Still, this algorithmic countermeasure has great worth in low-cost devices, such as smart cards and small hardware implementations, because it is possible to construct a powerful countermeasure against HODPA by combining the masking method with another countermeasure, such as clock randomization and the insertion of noise and random delays.

1. Motivation

Because of these merits, the masked implementation for block ciphers such as ARIA and AES has been the subject of a great deal of active research in terms of both software and hardware implementation. When we construct the masked

Manuscript received Mar. 17, 2009; revised Oct. 19, 2009; accepted Nov. 3, 2009.

This work was supported in part by the Second Brain Korea 21 Project and by the research program 2010 of Kookmin University, Rep. of Korea.

HeeSeok Kim (phone: +82 10 4749 3425, email: 80khs@korea.ac.kr) and Seokhie Hong (email: hsh@cist.korea.ac.kr) are with the Center for Information Security Technologies (CIST), Korea University, Seoul, Rep. of Korea.

Tae Hyun Kim (email: thkim7042@gmail.com) is with the Institute Attached to ETRI, Daejeon, Rep. of Korea.

Dong-Guk Han (corresponding author, email: christa@kookmin.ac.kr) is with the Department of Mathematics, Kookmin University, Seoul, Rep. of Korea.

doi:10.4218/etrij.10.0109.0181

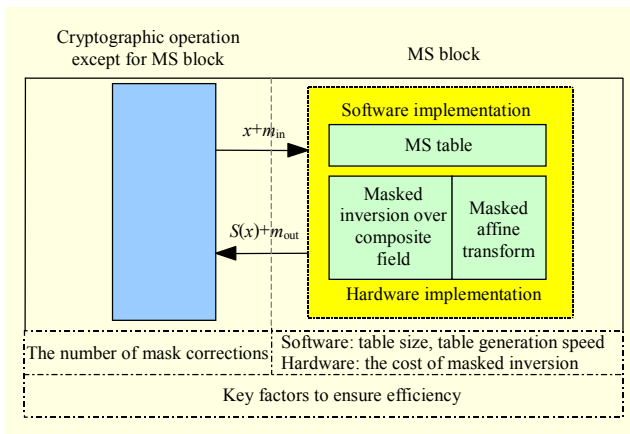


Fig. 1. Masked ARIA and AES.

implementation, mask corrections are inevitably required to produce correct output. As seen in Fig. 1, enhancing the efficiency of the entire scheme by minimizing the number of these mask corrections within the range that provides the security is the principal goal of both the masked hardware implementation and the masked software implementation.

However, the masked S-box (MS) block ($MS(x+m) = S(x)+m'$, where S , m , and m' are the original S-box, the input mask, and the output mask, respectively) is very different corresponding to implementations as seen in Fig. 1. In a masked software implementation, the MS table has been generated before en/decryption, and the S-box table of the original cipher is replaced by the MS table. The challenge is in how best to generate and use this MS table [7]–[9]. When the software implementation uses an MS table generated before the encryption or the decryption, both ARIA with four S-boxes and AES, including en/decryption, with 2 S-boxes may have a restriction related to the table size and a sharp drop in efficiency. Therefore, one must pay careful attention to the generation of the MS table.

By contrast, the masked hardware implementation computes the operation corresponding to the MS block directly. In order to find the most efficient method of carrying out this operation, there has been a great deal of research into computation methods (especially masked inversion) that do not use the MS table. Examples of this research can be found in [10]–[13]. A primary masked inversion uses the multiplicative masking method [10]. This method can be applied easily, but is susceptible to a zero-value (ZV) attack [3], in which an attacker can easily deduce the relationship between masked and unmasked values. Therefore, recent masked inversions have used an additive masking method over a composite field in order to achieve both security and efficiency [12], [13]. In the additive masking method, all the intermediate values must be blinded by adding random mask values. Various approaches to

additive masking for the inversion over a composite field have been proposed since Oswald first introduced this type. Although these methods are secure at the algorithmic level, they still have some weakness caused by glitches in the gate transition and the switching characteristics of XOR gates in masked multipliers [14]. However, these weaknesses can be easily eliminated by inserting delay elements into the paths of the input signals of the XOR gate, and so the additive masked inversion still has great value as a FODPA countermeasure.

2. Our Contribution

Our paper makes two contributions:

First, we propose an efficient masking structure that minimizes the number of mask corrections after constructing the MS block as shown in Fig. 1. Our masking structure for block ciphers such as ARIA and AES can be applied to both software and hardware implementations when the MS block has been well constructed in each implementation. When we consider additional operations besides those of the MS block, our masking structure needs only one additional 8-byte logical XOR operation for each round. The overhead time of this additional operation is only about 5.8% in the software implementation and 1 T_x (the time delay caused by one XOR gate) in the hardware implementation.

Second, we propose an efficient operation corresponding to the MS block in Fig. 1. In the software implementation, we propose a method that can reduce the table size of the MS table, which is the most significant part in the masked software implementation for ARIA and AES. In the case of ARIA with 4 S-boxes, the masking method needs 1,024 bytes of ROM and 1,024 bytes of RAM when a general method for generating the MS table is used. In the case of AES including en/decryption, the requirements are 512 bytes of ROM and 256 bytes of RAM [7]. In order to solve this table-size problem, we propose a method to generate the MS table efficiently. This method can reduce the ROM requirement by 50% to 512 bytes of ROM in ARIA and 256 bytes of ROM in AES. Furthermore, this method can reduce the clock cycles needed in ARIA because of the memory access reduction. In the hardware implementation, we also propose an efficient masked inversion method in the MS block. Among existing masked inversions known at present, Zakeri's method using normal bases is known to be optimal [13]. We offer a new efficient masked inversion by eliminating common expressions and finding new Boolean expressions. Our method can reduce the number of gates by about 10.5% in comparison with Zakeri's method.

The remainder of this paper is organized as follows. In section II, we describe the block ciphers ARIA and AES. In section III, we explain existing masking methods. Section IV

describes our proposed masking method and its efficiency. In section V, we demonstrate the security of our method. Finally, in section VI, we offer a conclusion.

II. Block Ciphers ARIA and AES

ARIA is a block cipher designed in 2003 by South Korean researchers; in 2004, the Korean Agency for Technology and Standards selected it as a standard cryptographic technique [15]. By contrast, the Advanced Encryption Standard (AES), also known as Rijndael, is a block cipher adopted as an encryption standard by the US government [16]. These two block ciphers are both composed of an SPN structure [17], [18].

For an N -bit SPN-type block cipher of r rounds, each round consists of three layers. These layers are the key mixing layer, the substitution layer, and the linear transformation layer. In the key mixing layer, the round input is bitwise XORed with the subkey for each round. In the substitution layer, the value resulting from the key mixing layer is partitioned into N/n blocks of n bits and each block of n bits then outputs other n bits through the bijective mapping from $\{0, 1\}^n$ to $\{0, 1\}^n$. In the case of ARIA and AES, an S-box fulfills the role of this layer. ARIA uses 2 S-Boxes, S_1 and S_2 , and their inverses, S_1^{-1} and S_2^{-1} , in the decryption scheme as well as in the encryption scheme. AES uses 1 S-box, S_1 , in the encryption scheme and uses its inverse, S_1^{-1} , in the decryption scheme. Each S-box is defined by an affine transformation and the inverse mapping over $GF(2^8)$ as in the following equations.

$$S_1, S_2 : GF(2^8) \rightarrow GF(2^8),$$

$$S_1(x) = Bx^{-1} \oplus b,$$

$$S_2(x) = CDx^{-1} \oplus c,$$

where B and CD are 8×8 matrices; and b and c are 8×1 vectors.

The resulting value of the substitution layer becomes the N -bit input $(i_0, i_1, \dots, i_{N-1})$ of the linear transformation layer. The linear transformation layer consists of multiplication by the $N \times N$ matrix. That is, the resulting value of this layer is $O = CI$, where C is an $N \times N$ matrix and I is $(i_0, i_1, \dots, i_{N-1})^T$. While the Diffusion layer fulfills this role in the ARIA, ShiftRows and MixColumns play this role in the AES. The linear transformation layer is usually omitted from the last round since it is easily shown that its inclusion adds no cryptographic strength.

III. Preliminaries

1. Masking

In a masked implementation, all intermediate values are

concealed by a random mask, which serves as a FODPA countermeasure. The masking method generates a random mask m and computes the masked ciphertext $y' (= y \oplus m')$ from the masked plaintext $x \oplus m$ for the plaintext x . Then the masking method computes $y' \oplus m'$ for obtaining the ciphertext y corresponding to the message x (the way this masking is applied depends on the cryptographic algorithm). This prevents a FODPA because the random mask values make it impossible for an attacker to predict intermediate values.

In masking method, we must know m' in order to get the ciphertext y . However, because of nonlinear parts, the block cipher has m' values that vary according to the different x values. If we try to determine a value m_x (the m' value corresponding to x) with no exposure of the intermediate values, this requires a significant amount of operations. Namely, when designing the masking method, we must consider these nonlinear parts in priority. The S-box operation, which is well known as the non-linear part of ARIA and AES, outputs $S(a) \oplus m_a$ from the input value $a \oplus m$. Because the m_a value is not linear with respect to a , we must make this output masking value the same for each a value. Although it is possible to compute this m_a value without the exposure of intermediate values, this method requires memory to keep track of all mask values as well as a large number of operations.

In order to fulfill this function, the software implementation has generally used the MS table. The MS table is generated by computing $MS(x \oplus m) = S(x) \oplus m'$ with new random numbers m , m' before the encryption or the decryption. This method requires only one additional MS table for each S-box. On the other hand, this method is not a good choice because of the register size in the hardware implementation. This problem requires the masking scheme in the hardware implementation to compute the output value corresponding to the MS table directly, without exposing the intermediate values.

2. The Inversion Operation over a Composite Field

In order to reduce the cost of the inversion in an S-box operation, an inversion method over a composite field has been proposed [18]. This method transforms an element over $GF(2^8)$ into an element over the composite field $GF(((2^2)^2)^2)$ having low inversion cost, and the inversion operation is actually carried out over this composite field. Then, the inversion operation over $GF(2^8)$ is completed by carrying out the inverse mapping into the element over $GF(2^8)$. We first define δ and δ^{-1} as the isomorphism functions between the two fields $GF(2^8)$ and $GF(((2^2)^2)^2)$. Two isomorphism functions are as follows:

$$\delta : GF(2^8) \rightarrow GF(((2^2)^2)^2),$$

$$\delta^{-1} : GF(((2^2)^2)^2) \rightarrow GF(2^8),$$

$$\delta = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}_2, \quad \delta^{-1} = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}_2.$$

The inversion over $GF(2^8)$ is computed by

$$x^{-1} \text{ over } GF(2^8) = \delta^{-1}[(\delta[x])^{-1} \text{ over } GF(((2^2)^2)^2)].$$

The inversion operation over the composite field is carried out over the subfield. The form of the irreducible polynomials used in each subfield is as follows:

$$GF(2^2) \text{ over } GF(2) : P_0(x) = x^2 + x + 1,$$

$$GF((2^2)^2) \text{ over } GF(2^2) : P_1(x) = x^2 + x + \phi,$$

$$GF(((2^2)^2)^2) \text{ over } GF((2^2)^2) : P_2(x) = x^2 + x + \lambda.$$

Let us define α , β , and γ as the roots of $P_0(x)$, $P_1(x)$, and $P_2(x)$, respectively. Then we can represent elements over $GF(2^2)$, $GF((2^2)^2)$, and $GF(((2^2)^2)^2)$ as $a_1\alpha + a_0$, $(a_1\alpha + a_2)\beta + (a_1\alpha + a_0)$, $\{(a_7\alpha + a_6)\beta + (a_5\alpha + a_4)\}\gamma + (a_3\alpha + a_2)\beta + (a_1\alpha + a_0)$, respectively. In order to reduce the inversion cost, ϕ over $GF(2^2)$ and λ over $GF((2^2)^2)$ are generally selected as α $((10)_2)$ and $(\alpha+1)\beta$ $((1100)_2)$, respectively.

The inversion operation over the composite field is performed as follows [19].

First, A^{-1} is computed by $C^{-1}A^{16}$ ($A = a_h\gamma + a_l \in GF(((2^2)^2)^2)$, $C = A^{17} \in GF((2^2)^2)$). Also, C^{-1} is computed by $D^{-1}C^4$ ($D = C^5 \in GF(2^2)$). That is, the inversion operation over $GF(((2^2)^2)^2)$ can be completed over $GF(2^2)$. This computation method unavoidably requires the operations of A^{16} and A^{17} , but A^{16} can be computed simply with only 4 bitwise XOR operations of $a_h\gamma + (a_h + a_l)$. Furthermore, A^{17} is computed simply by $a_h^2\lambda + (a_h + a_l)a_l$. Figure 2 represents the inversion operation over the composite field $GF(((2^2)^2)^2)$. For additional operations over both fields $GF((2^2)^2)$ and $GF(2^2)$, refer to [19].

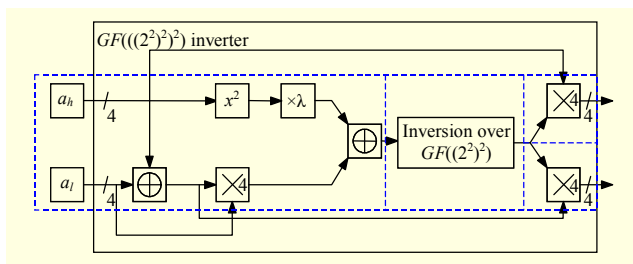


Fig. 2. Inverter over $GF(((2^2)^2)^2)$.

In Fig. 2, an equation for computing the inverse $A^{-1} = a_h'\gamma + a_l'$ of $A = a_h\gamma + a_l$ is performed as follows:

$$\text{Step 1. } d = \lambda a_h^2 + a_l(a_h + a_l).$$

$$\text{Step 2. } d' = d^{-1}.$$

$$\text{Step 3. } a_h' = d'a_h.$$

$$\text{Step 4. } a_l' = d'(a_h + a_l).$$

3. The Existing Masked Inversion Methods

In this subsection, we introduce the existing masked inversion methods over the composite field.

A. Akkar's Method

Akkar proposed the multiplicative masking method [10]. The method computes the following operation using the random number y for getting $x^{-1} \oplus m$ from $x \oplus m$.

$$x^{-1} \oplus m = (((x \oplus m)y \oplus my)^{-1} \oplus my^{-1})y.$$

However, this method needs a considerable number of additional operations and has the weakness of multiplicative masking. In the hardware implementation, we suppose that the optimized multiplication over $GF(2^8)$ needs 3 multiplications over $GF(2^4)$. Because Akkar's method needs 4 multiplications and 2 inversions over $GF(2^8)$, and 1 inversion over the composite field requires 3 multiplications over $GF(2^4)$, this method needs $4 \times 3 + 2 \times 3 = 18$ $GF(2^4)$ multiplications.

B. Blömer's Method

Blömer proposed an additive masking method for multiplication [11]. This method computes an output value $ab \oplus t$ (t : output masking) from two input values $a' = a \oplus r$, $b' = b \oplus s$ (r , s : input masking values) by $t \oplus a'b' \oplus rb' \oplus sa' \oplus rs$. However, this operation needs 4 $GF(2^8)$ multiplications for each masking multiplication. Therefore, this method needs a total of $4 \times 3 = 12$ $GF(2^4)$ multiplications.

C. Oswald's Method and Zakeri's Method

Oswald's method [12] and Zakeri's method [13] are additive masking schemes for the entire structure of the inversion operation over the composite field. These methods reduce the cost by finding and eliminating duplicate equations of $GF((2^2)^2)$ multiplications. The difference between the two methods lies in the selection of the basis. While Oswald selected a polynomial basis, Zakeri selected a normal basis. Both methods need 8 $GF(2^4)$ multiplications, but Oswald's method needs more of the additional operations, such as squaring, than Zakeri's method. These methods are known to

be the optimal versions among existing masked inversion methods.

IV. High-Speed and Low-Area Masking Method

The memory requirements and the time of the masking method are dependent on additional operations and table sizes. In the case of ARIA and AES, if a designer uses a general masking scheme in a software implementation, ARIA requires an additional 4 MS tables, and AES needs an additional 1 MS table. Although the operation corresponding to the MS block is computed directly in the hardware implementation, we can expect that this part will require numerous additional operations. In this section, we introduce the entire structure of the proposed masking scheme in which additional operations are seldom required, and we construct a masking structure for the S-box (MS block) in two versions of the software and hardware architecture.

1. Structure of the Proposed Masking Scheme

In this subsection, we explain the entire structure of the proposed masking method that is appropriate for the block cipher ARIA: the part corresponding to all the cryptographic operations in Fig. 1 except for the MS block. We do not consider the masking scheme for the entire structure of AES because it is possible for the method used for ARIA to be applied to AES as well. In the case of AES, we consider only the efficient structure for the MS block.

Initially, our proposed scheme for ARIA progresses as follows:

- Generate two 1-byte random masks, m and m' , and then compute $m'' (=m \oplus m')$.
- In order to satisfy the following equations for S_1 , S_2 , S_1^{-1} , and S_2^{-1} , generate MS tables in the software implementation or compute the MS block in the hardware implementation.

$$\begin{aligned} MS_1(x \oplus m) &= S_1(x) \oplus m', \\ MS_2(x \oplus m) &= S_2(x) \oplus m, \\ MS_1^{-1}(x \oplus m') &= S_1^{-1}(x) \oplus m, \\ MS_2^{-1}(x \oplus m') &= S_2^{-1}(x) \oplus m. \end{aligned} \quad (1)$$

We will introduce a method for generating MS tables and a method for computing the MS block in the following subsections.

- Blind all 16-byte round keys by the mask value $(m'')^{16}$ where $(m'')^{16}$ is $(=m''m''m''m''/m''m''m''m''/m''m''m''m''/m''m''m''m'')$.
- Blind the 16-byte plaintext by $m'm'm'm/m'm'm'm/m'm'm'm/m'm'm'm$.
- In each round, change the 16-byte output value of the

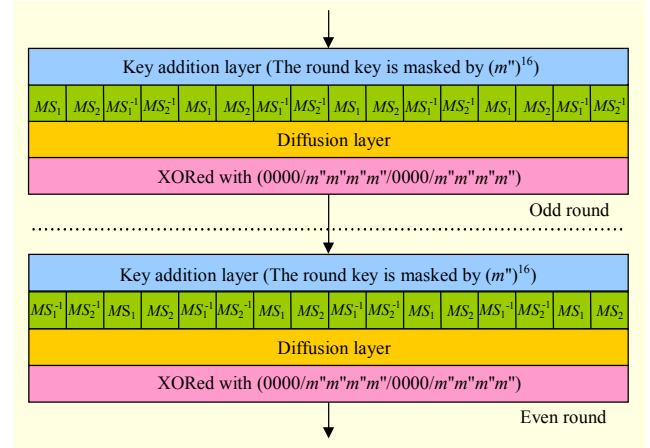


Fig. 3. Two-round masking structure of ARIA.

diffusion layer into the value XORed with $0000/m''m''m''m''/0000/m''m''m''m''$.

- Eliminate the masking value of the ciphertext by XORing with $m'm'm'm/m'm'm'm/m'm'm'm/m'm'm'm$.

Figure 3 shows the two-round structure of our masking scheme. In the odd rounds, an input value is masked by $m'm'm'm/m'm'm'm/m'm'm'm/m'm'm'm$. Because the masking value of the subkey is $(m'')^{16}$, the masking value of the intermediate result becomes $mmm'm'/mmm'm'/mmm'm'/mmm'm'$ after the key addition. After computing the MS block, the masking value before the diffusion layer becomes $m'm'm'm/m'm'm'm/m'm'm'm/m'm'm'm$. The diffusion layer of ARIA computes each output byte by EXORing (bitwise addition modulo 2) 7 of the 16 input bytes $I=(I_0, I_1, \dots, I_{15})^T$. Strictly speaking, this layer consists of multiplication by a 16×16 matrix. That is, the resulting value of this layer is $O=CI$, where

$$C = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

Because of this characteristic of the diffusion layer, the output mask of this layer becomes $mmm'm'/m'm'm'm/mmm'm'/$

Table 1. Time complexity of our masking method.

	ARIA without masking	ARIA with masking
Clock cycle	6,624 cc	7,009 cc

$m'm'mm$. This time, a 7-byte logical XOR operation gets accomplished with 4 bytes (x_0, x_1, x_2, x_3) masked by m (or m') and 3 bytes (y_0, y_1, y_2) masked by m' (or m), and so we must adjust the order of XOR operations in order to not expose intermediate values (for example, $((x_0 \oplus y_0) \oplus x_1) \oplus (y_1 \oplus x_2) \oplus (y_2 \oplus x_3)$). As can be seen in Fig. 3, we must adjust the masking value by an 8-byte logical XOR operation after the diffusion layer. Except for an additional operation of the MS block, this operation is the only additional operation needed in our masking scheme.

The masking scheme for even rounds uses a similar method.

For the software implementation, the clock cycles needed for encryption or decryption, excluding the parts generating the MS tables, are shown in Table 1. This numerical value was measured in an ATmega128L [20].

As seen in Table 1, our method needs only 5.8% more time to perform the 1 additional 8-byte logical XOR operation per round. The additional operation results in a time delay of only 1 T_X in the hardware implementation.

2. The Design of the Masking S-box

A. Method to Generate the MS Tables in Software

In the software implementation, the masking method usually uses a method that generates an MS table before en/decryption. In general, one S-box requires one MS table (General method) [21], [22]. In particular, the masking method for ARIA needs 1 kB ROM for four S-boxes and 1 kB RAM for four MS tables. The masking method for AES including en/decryption requires 512 bytes of ROM and 256 bytes of RAM. In order to reduce the ROM size, ARIA and AES can only store forward S-boxes. In this case, one will compute its inverse in RAM and then mask this version in RAM. Although these methods can reduce the ROM size by 50%, this method (Method A) requires more time.

We propose a simple but effective idea to partially solve this table size problem without affecting the performance; this fact can increase the efficiency of the masking scheme in software as depicted in Fig. 1.

ARIA. In the case of ARIA, our method stores two S-boxes (S_1, S_2) in ROM and then generates four MS tables ($MS_1, MS_2, MS_1^{-1}, MS_2^{-1}$) satisfying (1) from these two S-boxes. Algorithm 1 provides the method for generating MS tables.

Algorithm 1. Proposed method generating MS tables.

Input: S_1, S_2, m, m' .

Output: $MS_1, MS_2, MS_1^{-1}, MS_2^{-1}$.

For $x=0x00$ to $0xff$ do

$u=x \oplus m, v=S_1(x) \oplus m', w=S_2(x) \oplus m'$.

$MS_1(u) = v, MS_2(u) = w, MS_1^{-1}(v) = u, MS_2^{-1}(w) = u$.

Return $MS_1, MS_2, MS_1^{-1}, MS_2^{-1}$.

In the four MS tables generated by algorithm 1, the first two subequations of (1) are trivially true. The two other subequations corresponding to $MS_1^{-1}(x \oplus m') = S_1^{-1}(x) \oplus m$, $MS_2^{-1}(x \oplus m') = S_2^{-1}(x) \oplus m$ are satisfied by proposition 1.

Proposition 1. If $MS_1^{-1}(S_1(x) \oplus m')$ is equal to $x \oplus m$, $MS_1^{-1}(x \oplus m') = S_1^{-1}(x) \oplus m$ is satisfied.

Proof. Because S_1 is a permutation, for any x there exists one, and only one t , such that $x = S_1^{-1}(t)$. If we substitute x for $S_1^{-1}(t)$, the equation $MS_1^{-1}(S_1(x) \oplus m') = x \oplus m$ can be transformed into $MS_1^{-1}(t \oplus m') = S_1^{-1}(t) \oplus m$. \square

When our method is compared with the general method for generating MS tables, our method can result in a reduction of 0.5 kB in ROM because it needs 0.5 kB ROM and 1 kB RAM. The clock cycles needed for generating MS tables are also reduced, as seen in Table 2, because of the memory access reduction. Also, when our method is compared with Method A, the clock cycles are reduced. These numerical values were measured in an ATmega128L.

AES including en/decryption. In the case of AES including en/decryption scheme, our proposed method does not need an inverse S-box S_1^{-1} . In the decryption scheme of AES, when we generate the MS table MS_1^{-1} of S_1^{-1} , we can get this table using only S_1 by using the equation $MS_1^{-1}(S_1(x) \oplus m') = x \oplus m$. This method is similar to ARIA.

Under the same conditions given for ARIA above, the cost

Table 2. Cost of generating MS tables in ARIA.

	General method	Method A	Our method
Clock cycle	11,315 cc	17,112 cc	9,752 cc
ROM	1,024 bytes	512 bytes	512 bytes
RAM	1,024 bytes	1,024 bytes	1,024 bytes

Table 3. Cost of generating MS tables in AES including en/decryption.

	General method	Method A	Our method
Clock cycle	5,658 cc	11,312 cc	5,658 cc
ROM	512 bytes	256 bytes	256 bytes
RAM	256 bytes	256 bytes	256 bytes

for an MS table of AES are shown in Table 3.

B. The Structure of the MS Block in Hardware

In the hardware implementation of ARIA and AES, the structure of the MS block is the most significant part of the entire structure. In this subsection, we propose an efficient structure of the MS block appropriate for the hardware implementation: the part corresponding to the hardware implementation of the MS block in Fig. 1. In ARIA and AES, the S-box is comprised of the inversion operation and the affine transform. Because of the inversion corresponding to a non-linear operation, the masking method for S-boxes always requires great expense. In order to reduce this expense, we use only one masked inversion.

ARIA. In the case of ARIA, the detailed structure of the MS block is schematically depicted in Fig. 4. Table 4 presents the input/output values corresponding to a select signal s .

AES including en/decryption. In the case of AES, the structure of the MS block is only slightly different from ARIA. This can be accomplished by eliminating signals 2 and 4 in Fig. 4, and so Fig. 4 can be applied to the unified structure of ARIA and AES [23].

Masked inversion block. The only remaining part of our method is the masked inversion block of the MS block.

In Fig. 4, the masked inversion block has 3 8-bit input values $(a_h \oplus m_h \parallel a_l \oplus m_l)$, $(m_h \parallel m_l)$, $(m_h' \parallel m_l')$ and then computes an 8-bit output value $(a_h' \oplus m_h' \parallel a_l' \oplus m_l')$.

We define the following symbols before introducing our proposed method.

- A : unmasked input value of $GF((2^2)^2)$ inverter ($A = (a_h \parallel a_l)$, $a_h, a_l \in GF((2^2)^2)$).
- A^{-1} : unmasked output value of $GF((2^2)^2)$ inverter ($A^{-1} = (a_h' \parallel a_l')$, $a_h', a_l' \in GF((2^2)^2)$).
- m : masking value of A ($m = (m_h \parallel m_l)$, $m_h, m_l \in GF((2^2)^2)$).
- m' : masking value of A^{-1} ($m' = (m_h' \parallel m_l')$, $m_h', m_l' \in GF((2^2)^2)$).
- d : unmasked input value of $GF((2^2)^2)$ inverter.
- d' : unmasked output value of $GF((2^2)^2)$ inverter.
- m_d : masking value of d .
- $A \oplus m = (\tilde{a}_h \parallel \tilde{a}_l) = (a_h \oplus m_h \parallel a_l \oplus m_l)$.
- $\tilde{d}' = d' \oplus m_d$.
- $A^{-1} \oplus m' = (\tilde{a}_h' \parallel \tilde{a}_l') = (a_h' \oplus m_h' \parallel a_l' \oplus m_l')$.

Our proposed method uses the masked inversion over the composite field. The method is accomplished with the following four steps.

Step 1. Compute $d \oplus m_d$ from $(a_h \oplus m_h \parallel a_l \oplus m_l)$.

Step 2. Compute $d' \oplus m_h$ from $d \oplus m_d$.

Step 3. Compute $\tilde{a}_h' (= d' \oplus a_h \oplus m_h)$ from $d' \oplus m_h$ and $a_h \oplus m_h$.

Step 4. Compute $\tilde{a}_l' (= d' \oplus a_l \oplus m_l)$ from $d' \oplus m_h$, $a_h \oplus m_h$ and $a_l \oplus m_l$.

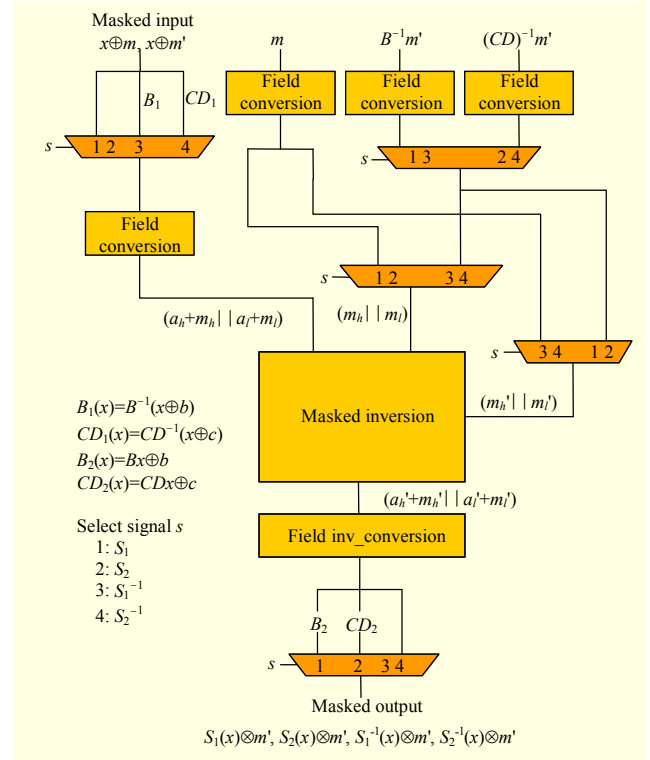


Fig. 4. Structure of the MS block.

Table 4. Input/output values corresponding to a select signal.

Select signal s	Input values	Output value
1	$x \oplus m, m, m'$	$S_1(x) \oplus m'$
2	$x \oplus m, m, m'$	$S_2(x) \oplus m'$
3	$x \oplus m', m, m'$	$S_1^{-1}(x) \oplus m$
4	$x \oplus m', m, m'$	$S_2^{-1}(x) \oplus m$

Each step carries out the procedure for a secure computation of corresponding blocks in Fig. 2. All the intermediate values in the computation of each step must be independent of all values which can be generated by A ($= (a_h \parallel a_l)$) and the constant value λ . We don't consider Step 2 corresponding to the masked inversion over $GF((2^2)^2)$ because this computation is very similar to that over $GF(((2^2)^2)^2)$ [12], [13].

Step 1. Computation of $d \oplus m_d$

$d \oplus m_d (= a_h^2 \lambda \oplus a_l(a_h \oplus a_l) \oplus m_d)$ can be computed by the following equation.

$$\underbrace{\lambda \tilde{a}_h^2}_a \oplus \underbrace{m_h(\tilde{a}_h \oplus \tilde{a}_l)}_{M1} \oplus \underbrace{m_d}_{M2} \oplus \underbrace{\tilde{a}_l(\tilde{a}_h \oplus \tilde{a}_l)}_b \oplus \underbrace{m_h m_l}_{M2} \oplus \underbrace{\tilde{a}_h(m_h \oplus m_l)}_{X3} \oplus \underbrace{m_l^2}_{S1} \oplus \underbrace{\lambda m_h^2}_{S1}. \quad (2)$$

The security for each multiplication, square, and multiplication with the constant (CM), is mentioned in the next

section. In (2), we must decide on the order of XOR operations on the results of multiplications, squares, and CMs. In the case that any relation with the expected intermediate values is originated inevitably, we must eliminate this relation by using a fresh_mask (fm). If we adjust the order of (2) using fm , (2) can be converted into

$$(((fm \oplus a) \oplus M1) \oplus ((m_d \oplus b) \oplus M2)) \oplus ((fm \oplus \lambda S1) \oplus X3). \quad (3)$$

Equation (3) is schematically depicted in Fig. 5(a).

Step 3. Computation of $\tilde{a}_h' (=d'a_h \oplus m_h')$.

$d'a_h \oplus m_h'$ can be computed by

$$m_h' \oplus X3 \oplus \underbrace{(\tilde{a}_h \oplus (m_h \oplus m_i))}_{M4} \oplus \underbrace{(\tilde{d}' \oplus m_i)}_{X4} \oplus m_i' \oplus \tilde{d}' m_i \oplus m_i' S1 \oplus M2. \quad (4)$$

In (4), the reason for adding m_i' twice is similar to that for using fm , that is, because results of each addition must have no relation with any of the expected intermediate values. If we adjust the order of (4) for security, (4) can be converted into

$$(((m_h' \oplus X3) \oplus M4) \oplus m_i') \oplus ((\tilde{d}' m_i \oplus m_i') \oplus (S1 \oplus M2)). \quad (5)$$

Step 4. Computation of $\tilde{a}_i' (=d'(a_h \oplus a_i) \oplus m_i')$.

$d'(a_h \oplus a_i) \oplus m_i'$ can be computed by the following equation.

$$(\tilde{a}_h \oplus \tilde{a}_i \oplus m_h) \tilde{d}' \oplus M1 \oplus X4. \quad (6)$$

If we adjust the operation order for security, (6) can be converted into

$$((\tilde{a}_h \oplus \tilde{a}_i \oplus m_h) \tilde{d}' \oplus M1) \oplus X4. \quad (7)$$

Equations (5) and (7) are schematically depicted in Fig. 5 (b).

In our proposed method, fm and m_d are 4-bit random numbers, but these values can be selected as $fm=m_h'$ and $m_d=m_i'$ in order to reduce the cost of random number generation.

Our masked inversion method can reduce the area complexity with respect to existing methods [10]-[13] by eliminating common expressions and finding new Boolean expressions. Table 5 presents the comparison between our method and existing methods, and it lists the numbers of high-level operations (multiplication, square, CM over $GF((2^2)^2)$). We don't count the numbers of additions over $GF((2^2)^2)$ because the cost is relatively small and the number is similar for all the schemes.

Multiplication needs about 5.5 times more gates than squaring and about 7.5 times more than CM [24]. Thus our method is a 10.5% improvement over Zakeri's method in terms of area complexity.

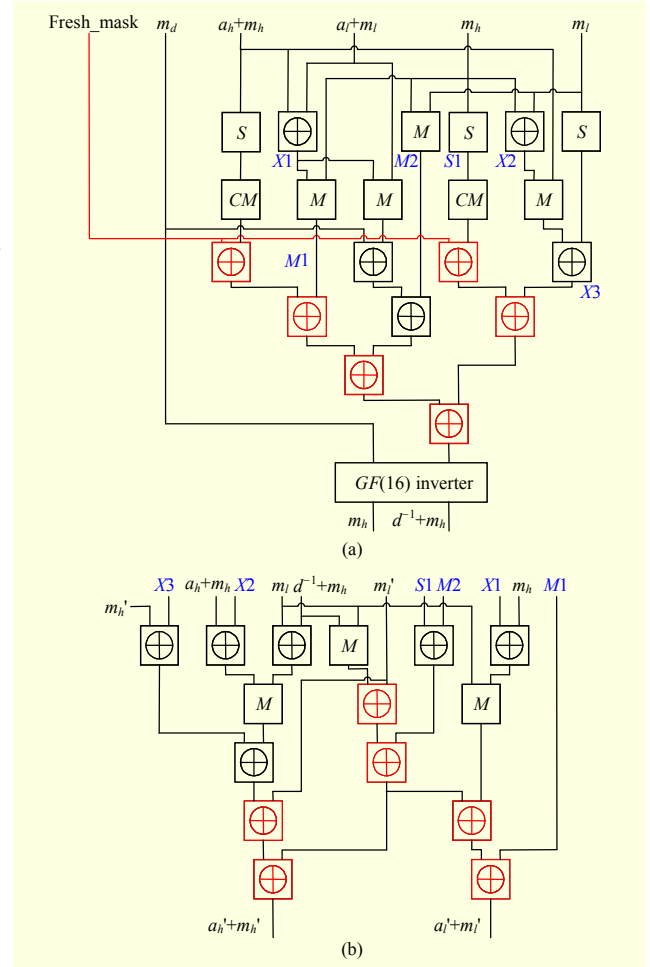


Fig. 5. Schematic of the masked inversion.

Table 5. Comparison of masked inversion.

Method	Multiplication	Square	CM
Akkar [10]	18	6	4
Blömer [11]	12	1	2
Oswald [12]	8	4	2
Zakeri [13]	8	2	2
Our method	7	3	2

V. Security

1. The Security of the Software Implementation

The security of our masking method in the software implementation can be proven with lemma 1 and lemma 2. As the proofs of these lemmas are straightforward, we omit them.

Lemma 1. Let an element a of $GF(2^n)$ be arbitrary. Let m be uniformly distributed in $GF(2^n)$ and independent of a . Then the distribution of $a \oplus m$ is independent of a .

Lemma 2. Let an element a of $GF(2^n)$ be arbitrary. Let m_1 and m_2 be uniformly distributed in $GF(2^n)$ and independent of a . Then the distribution of $a \oplus m_1 \oplus m_2$ is independent of a .

These lemmas describe the basic reasons that additive masking is secure. Namely, if all intermediate values which can be attacked are concealed by random masks independent of these values, it is completely impossible for an attacker to predict intermediate values. Therefore, the use of the additive masking with the mask uniformly distributed prevents FODPA.

2. The Security of the Hardware Implementation

The security of our design in the hardware implementation can be proven by lemma 1, lemma 2, lemma 3, lemma 4, and lemma 5. For a proof of each lemma, one can refer to [12].

Lemma 3. Let elements a and b of $GF(2^n)$ be arbitrary. Let m_a and m_b be independently and uniformly distributed in $GF(2^n)$. Then the distribution of $(a \oplus m_a)(b \oplus m_b)$ is independent of a and b .

Lemma 4. Let an element a of $GF(2^n)$ be arbitrary. Let m_a and m_b be independently and uniformly distributed in $GF(2^n)$. Then the distribution of $(a \oplus m_a)m_b$ is independent of a .

Lemma 5. Let an element a of $GF(2^n)$ be arbitrary and λ be a constant. Let m_a be independently and uniformly distributed in $GF(2^n)$. Then, the distribution of $(a \oplus m_a)^2$ and $(a \oplus m_a)^2 \lambda$ is independent of a .

Lemma 3 and lemma 4 provide the form of input values of multipliers for constructing secure inverters. Namely, two input values of the multiplier must use different masking values that are not related. Finally, lemma 5 implies that the output value of squaring and CM is also secure against a FODPA when input values are securely masked.

In the hardware implementation, our masked inversion has been constructed using only the above lemmas. Therefore, the security of our method is proven.

VI. Conclusion

In this paper, we have proposed a new masking method appropriate for block ciphers ARIA and AES. In the software implementation we designed a new method generating an MS table that can reduce the ROM size of tables. We also proposed a new masked inversion method, which is the main part of the MS block in the hardware implementation. Our MS block can reduce ROM size in the software implementation by 50% and the gates by more than 10.5% over Zakeri's method in the hardware implementation. Under the assumption that we have a secure MS block, we have designed the remaining part by minimizing the number of mask corrections. Additional operations corresponding to this part require no more than

8-byte logical XOR operations for each round.

References

- [1] P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," *Int. Conf. Cryptology*, 1999, pp. 388-397.
- [2] J. Ha et al., "Differential Power Analysis on Block Cipher ARIA," *HPCC, LNCS*, vol. 3726, 2005, pp. 541-548.
- [3] J.D. Golic and C. Tymen, "Multiplicative Masking and Power Analysis of AES," *CHES, LNCS*, vol. 2523, 2002, pp. 198-212.
- [4] C. Kim, M. Schl  ffer, and S. Moon, "Differential Side Channel Analysis Attacks on FPGA Implementations of ARIA," *ETRI J.*, vol. 30, no.2, Apr. 2008, pp. 315-325.
- [5] F.X. Standaert, S.B.   rs, and B. Preneel, "Power Analysis of an FPGA Implementation of Rijndael: Is Pipelining a DPA Countermeasure?" *CHES, LNCS*, vol. 3156, 2004, pp. 30-44.
- [6] T. Messerges, "Using Second-Order Power Analysis to Attack DPA Resistant Software," *CHES, LNCS*, vol. 1965, 2000, pp. 238-251.
- [7] T. Messerges, "Securing the AES Finalists Against Power Analysis Attacks," *FSE, LNCS*, vol. 1978, 2000, pp. 150-164.
- [8] E. Trichina, D.S. Seta, and L. Germani, "Simplified Adaptive Multiplicative Masking for AES," *CHES, LNCS*, vol. 2523, 2002, pp. 187-197.
- [9] K. Schramm and C. Paar, "Higher Order Masking of the AES," *LNCS*, vol. 3860, 2006, pp. 208-225.
- [10] M. L. Akkar and C. Giraud, "An Implementation of DES and AES, Secure Against Some Attacks," *CHES, LNCS*, vol. 2162, 2001, pp. 309-318.
- [11] J. Bl  mer, J. Guajardo, and V. Krummel, "Provably Secure Masking of AES," *SAC, LNCS*, vol. 3357, 2005, pp. 69-83.
- [12] E. Oswald et al., "A Side-Channel Analysis Resistant Description of the AES S-Box," *FSE, LNCS*, vol. 3557, 2005, pp. 413-423.
- [13] B. Zakeri et al., "Compact and Secure Design of Masked AES S-Box," *Lecture Notes in Computer Science*, vol. 4861, 2007, pp. 216-229.
- [14] S. Mangard, N. Pramstaller, and E. Oswald, "Successfully Attacking Masked AES Hardware Implementations," *CHES, LNCS*, vol. 3659, 2005, pp. 157-171.
- [15] D. Kwon et al., "New Block Cipher: ARIA," *ICISC, LNCS*, vol. 2971, 2004, pp. 432-445.
- [16] J. Daemen and V. Rijmen, *The Design of Rijndael: AES - The Advanced Encryption Standard*, Springer, 2002.
- [17] C. Adams and S. Tavares, "The Structured Design of Cryptographically Good SBoxes," *J. of Cryptology*, vol. 3, no. 1, 1990, pp. 27-42.
- [18] L. O'Connor, "On the Distribution of Characteristics in Bijective Mappings," *Eurocrypt, LNCS*, vol. 765, 1994, pp. 360-370.
- [19] A. Satoh et al., "A Compact Rijndael Hardware Architecture with S-Box Optimization," *ASIACRYPT, LNCS*, vol. 2248, 2001, pp.

239-254.

- [20] Atmel Corporation. Datasheet: ATmega128(L). <http://www.atmel.com/products/avr/>.
- [21] C. Herbst, E. Oswald, and S. Mangard, "An AES Smart Card Implementation Resistant to Power Analysis Attacks," *ACNS, LNCS*, vol. 3989, 2006, pp. 239-252.
- [22] E. Oswald and K. Schramm. "An Efficient Masking Scheme for AES Software Implementations," *WISA, LNCS*, vol. 3786, 2006, pp. 292-305.
- [23] B. Koo et al., "Design and Implementation of Unified Hardware for 128-Bit Block Ciphers ARIA and AES," *ETRI J.*, vol. 29, no. 6, Dec. 2007, pp. 80-82.
- [24] J. Wolkerstorfer, E. Oswald, and M. Lamberger, "An ASIC Implementation of the AES SBoxes," *CT-RSA, LNCS*, vol. 2271, 2002, pp. 67-78.

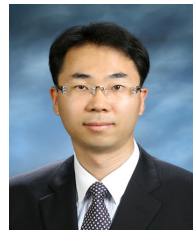


HeeSeok Kim received his BS degree in mathematics from Yonsei University, Seoul, Rep. of Korea, in 2006 and the MS degree in engineering in information security from Korea University, Seoul, Rep. of Korea in 2008. He is currently a PhD student with Korea University.

He is also a researcher with the Center for Information Security Technologies (CIST). His research interests include side channel attacks, public-key cryptosystems, and symmetric-key cryptosystems.



Tae Hyun Kim received the BS degree in mathematics from University of Seoul, Seoul, Rep. of Korea, in 2002 and the MS degree in engineering in information security from Korea University, Seoul, Rep. of Korea, in 2004. He was a visiting researcher with the School of Systems Information Science, Future University, Hakodate, Japan, from April to September 2006. He received his PhD in engineering in information security from Korea University, Seoul, Rep. of Korea, in 2009. Since August 2009, he has been a researcher with the Attached Institute of ETRI, Daejeon, Rep. of Korea.



Dong-Guk Han received his BS degree in mathematics from Korea University in 1999, and his MS degree in mathematics from Korea University in 2002. He received his PhD in engineering in information security from Korea University in 2005. He was a postdoctoral researcher at Future University, Hakodate,

Japan. After finishing the doctoral course, he was an exchange student in the Department of Computer Science and Communication Engineering in Kyushu University in Japan from April 2004 to March 2005. He was a senior researcher in ETRI, Daejeon, Korea. He is currently working as an assistant professor with the Department of Mathematics of Kookmin University, Seoul, Korea. He is a member of KIISC, IEEK, and IACR.



Seokhie Hong received his MA and PhD degrees in mathematics from Korea University in 1997 and 2001, respectively. He worked for SECURITY Technologies Inc. from 2000 to 2004. From 2004 to 2005, he worked as a postdoctoral researcher with COSIC at KU Leuven, Belgium. Since 2005, he has been with

Korea University, where he is now working in the Graduate School of Information Management and Security. His specialty lies in the area of information security, and his research interests include the design and analysis of symmetric-key cryptosystems, public-key cryptosystems, and forensic systems.