Secure Hardware Implementation of ARIA Based on Adaptive Random Masking Technique

Junki Kang, Dooho Choi, Yong-Je Choi, and Dong-Guk Han

The block cipher ARIA has been threatened by side-channel analysis, and much research on countermeasures of this attack has also been produced. However, studies on countermeasures of ARIA are focused on software implementation, and there are no reports about hardware designs and their performance evaluation. Therefore, this article presents an advanced masking algorithm which is strong against second-order differential power analysis (SODPA) and implements a secure ARIA hardware. As there is no comparable report, the proposed masking algorithm used in our hardware module is evaluated using a comparison result of software implementations. Furthermore, we implement the proposed algorithm in three types of hardware architectures and compare them. The smallest module is 10,740 gates in size and consumes an average of 47.47 µW in power consumption. Finally, we make ASIC chips with the proposed design, and then perform security verification. As a result, the proposed module is small, energy efficient, and secure against SODPA.

Keywords: ARIA, masking, side-channel analysis, low-power design, ASIC.

I. Introduction

1. Related Works

Side-channel analysis reveals secret values using physical information like operation times, electromagnetic radiation, and power consumption from cryptographic devices. Since Kocher introduced differential power analysis (DPA) in 1998 [1], much research about side-channel analysis and its countermeasures has been produced. Side-channel analysis attacks crypto-systems using the relationship between intermediate values in crypto devices and physical information measured from the devices. In other words, the attacks can be defended by breaking this relationship. The DPA attack, one of the most powerful side-channel analysis methods, can also be prevented by breaking correlation between intermediate values and measured power signals using masking or hiding methods [2].

Masking, one of the DPA countermeasures, inserts random values into the intermediate values in cipher calculations to break correlation, and the intermediate values in the crypto system are ambiguously changed. Masking methods efficiently prevent DPA attacks, but they are insecure against second-order DPA (SODPA) attacks [3]. Therefore, many researchers are looking into ways to defend against SODPA by applying additional algorithms like the shuffling method.

The block cipher ARIA has also been threatened by DPA attacks, and studies on the masking are underway to prevent such dangers. As masking algorithms can avoid only first-order DPA, the shuffling method is generally used to inhibit the SODPA attacks. However, it is hard to apply a masking algorithm including a shuffling method for the block cipher ARIA in computationally limited environments because the ARIA algorithm is large and slow compared with other block

Manuscript received Apr. 25, 2011; revised Aug. 25, 2011; accepted Sept. 6, 2011.

This work was supported by SCARF project which is the R&D program of KCC/KCA [Development of the Technology of Side Channel Attack Countermeasure Primitives and Security Validation]. This work was also supported by research program 2012 of Kookmin University in Korea.

Junki Kang (phone: +82 42 860 1729, kang.junki@gmail.com) is with the Cyber Security-Convergence Research Department, ETRI, Daejeon, Rep. of Korea, and is also with the University of Science and Technology, Daejeon, Rep. of Korea.

Dooho Choi (dhchoi@etri.re.kr) and Yong-Je Choi (choiyj@etri.re.kr) are with the Cyber Security-Convergence Research Department, ETRI, Daejeon, Rep. of Korea.

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

http://dx.doi.org/10.4218/etrij.12.0111.0251

ciphers like AES. Moreover, most research on ARIA countermeasures focus on software implementation. Thus, we propose a new ARIA masking algorithm appropriate for hardware designs and implement our proposed algorithm in ASIC chip.

2. Contributions

We present two developments in this article. First, we introduce an advanced ARIA masking algorithm named adaptive random masking. The other development is that we efficiently implement the above algorithm to various hardware architectures and compare them. Moreover, we implement our design in an ASIC chip and verify its security.

A. Adaptive Random Masking Algorithm

Adaptive random masking basically uses different random numbers for each input block in a round, and this work can defend the SODPA without additional protection like S-box shuffling. Of course, previous masking algorithms can change their masks for every block when the algorithms calculate masking S-boxes for different masks, but this method is very inefficient. Our proposed method, however, is designed such that different random masks are utilized in every block efficiently in hardware implementation.

B. Performance Evaluation of Proposed Algorithm

Because there is no comparable research about hardware implementation of ARIA masking, it is difficult to directly evaluate the proposed masking algorithm. Thus, we implement the adaptive random masking algorithm in software at first, and then we compare the proposed algorithm with other masking algorithms using the implementation result.

C. Hardware Implementation and Comparison

We implement the adaptive random masking algorithm to hardware, and show it is efficient for small-area and low-power devices. Because there are no comparable reports about masked ARIA hardware implementations, we implement our algorithm on three types of design and compare the performance themselves. As the result, the 8/16-bit architecture is the smallest and has approximately 7.1% areal overhead. Moreover, the power consumption in the 8/16-bit module has a 35.8% increment over the original ARIA module.

D. Security Evaluation

We experimentally evaluate the security of the proposed algorithm and its implementations. We program the software countermeasure into an 8-bit microcontroller and perform the SODPA resistant test. In the case of a hardware module, we evaluate the security using ASIC chips. In this way, we can experimentally verify that the adaptive random making method and its implementation are secure.

The remainder of this paper is organized as follows. We describe the block cipher ARIA and the previous studies about ARIA masking algorithms in section II. We introduce the adaptive random masking method in section III. In section IV, we evaluate the efficiency of the proposed algorithm with the software implementation results. Section V describes hardware implementations for the proposed masking algorithm and their performance. Verification of security is presented in section VI, and our conclusions are given in section VII.

II. Previous Works

1. Block Cipher ARIA

Block cipher ARIA, which is based on an involution SPN structure, encrypts and decrypts data in 128-bit blocks. ARIA can have a 128-bit, 192-bit, or 256-bit key size, and a corresponding number of round functions. A round operation in the ARIA procedure has a key addition, substitution layer, and diffusion layer. Round keys are composed by a combination of 512-bit expanded keys (W_0, W_1, W_2, W_3). In addition, the expanded key is made from round functions [4], [5].

The substitution layer of ARIA uses four S-boxes (S_1, S_2, S_1^{-1}, S_2^{-1}), and S_1^{-1} and S_2^{-1} are the inverses of S_1 and S_2. All the S-boxes in ARIA are defined over composite field $GF(2^8)$ with the irreducible polynomial $m(x) = x^8 + x^4 + x^3 + x + 1$, and the polynomial is identically used in AES algorithm. Thus, S-boxes in the ARIA can be described as

$$S_1(x) = A \cdot x^{-1} \oplus a,$$

$$S_2(x) = B \cdot x^{247} \oplus b = B \cdot x^{-8} \oplus b = D \cdot x^{-1} \oplus b,$$

$$S_1^{-1}(x) = (A^{-1} \cdot (x \oplus a))^{-1}, \qquad S_2^{-1}(x) = (D^{-1} \cdot (x \oplus b))^{-1},$$

Algorithm 1. ARIA algorithm

```
  Input: a plaintext $M$ with size $N_b$ and round key $w[N_b \times (N_r+1)]$
  Output: ciphertext C
  $state \leftarrow M$
  AddRoundKey(state, $w[0\ldots N_b-1]$)
  for $round$=1 to $N_r-1$ do
      SubstitutionLayer(state)
      DiffusionLayer(state)
      AddRoundKey(state, $w[round \times N_b, (round+1) \times N_b-1]$)
  end for
  SubstitutionLayer(state)
  DiffusionLayer(state)
  $C \leftarrow state$
  return $C$
```

where x^{-1} is the inversion function over $GF(2^8)$, and A, B, D, and D^{-1} are 8×8 binary matrices for affine transform. The matrix A and D are defined in [6]. The diffusion layer of ARIA is originally represented by a multiplication of the 16×16 involution matrix C with a 16×1 input vector $I=(I_0, I_1, \ldots, I_{15})^{\mathrm{T}}$; therefore, the diffusion output can be denoted as $O = C \cdot I$ [7].

Finally, the whole ARIA algorithm can be described as in algorithm 2, where $AddRoundKey(state, w[\ ]) = state \oplus w[\ ]$. In addition, N_b is denoted as the number of bytes and N_r is the number of rounds in algorithm 1.

2. Side-Channel Analysis on ARIA and Its Countermeasures

Since side-channel analysis was introduced, many crypto-systems have been threatened by this attack, including ARIA. The weakness of the ARIA algorithm against side-channel analysis was presented in various studies [8]. Both DPA and fault injection attacks can be performed on ARIA [9].

A. Previous Countermeasures for ARIA

The general masking algorithm and the algorithm using a masked inversion table are generally used as countermeasures for ARIA [10]. The general masking algorithm generates four masked S-boxes as shown in algorithm 2 for the masked substitution layer. This method requires 1,024 bytes of RAM and many times that to generate masked S-box tables, so it is inappropriate for hardware architecture. When the algorithm is implemented to software, total operation cycles for the general masking are about four times longer than original cipher's operation time, which is shown in Table 1.

Masking with a masked inversion table is relatively efficient for small memory environments as the algorithm requires only 256 bytes of RAM and 256 bytes of ROM. However, this method is very slow for calculation. This method is ten times slower than the original ARIA operation cycles. Moreover, when we change random masks at a round or an input block, the whole operation time is slowed down tremendously as shown in Table 1.

B. Motivation

Previous masking algorithms are inefficient for hardware design and embedded systems which have restrict resources.

```
Algorithm 2. Masked S-Boxes for general masking
  Input: S-boxes $(S_1, S_2, S_1^{-1}, S_2^{-1})$, $r$, $r'$
  Output: Masked S-boxes $MS_1, MS_2, MS_1^{-1}, MS_2^{-1}$
  for $i$=0 to 255 do
    $MS_1(i \oplus r) = S_1(i) \oplus r'$, $MS_2(i \oplus r) = S_2(i) \oplus r'$,
    $MS_1^{-1}(i \oplus r) = S_1^{-1}(i) \oplus r'$, $MS_2^{-1}(i \oplus r) = S_2^{-1}(i) \oplus r'$
  end for
  return $MS_1, MS_2, MS_1^{-1}, MS_2^{-1}$
```

Table 1. Comparison of ARIA with previous masking algorithm.

		ARIA	Masking with masked S-box tables	Masking with masked inversion
Resource	ROM	1,024	1,024	256
	RAM	-	1,024	256
Relative time to operate (clock cycle)	Full round operation	100% (7,920)	395% (31,257)	994% (78,723)
	Different mask for a block	-	4,379% (346,793)	2,512% (198,915)
	Different mask for a round	-	3,134% (248,188)	2,037% (161,355)

As the masking algorithms have low performance, many researchers are looking into ways to improve their efficiency. One of the ways is to reduce the number of calculations in masking table generation [7]. Applying random masks to only the first and final rounds in the cipher is also considered a fast method. These modified algorithms, however, still have security problems related to the SODPA attack, and these methods are not considered for hardware architecture and its efficiency. Thus, we consider that the proposed masking algorithm can be efficiently implemented to hardware and demonstrate that the proposed masking algorithm is strong against the SODPA attack using different random masks.

III. Adaptive Random Masking Algorithm for ARIA

Since the ARIA algorithm uses four S-boxes at the substitution layer, four masked S-boxes should be defined and, in general, the masked S-box table should be computed and stored at every round. However, in our proposed masking algorithm, we use only one masked inversion table, and the masked S-boxes are computed at each round when the algorithm is implemented to software. Furthermore, the proposed masking uses different random masks for every input block to defend against SODPA. For the proposed masking method, the inversion table and affine transform tables (A, D, A^{-1}, D^{-1}) are precomputed for operation speed in the case of software implementation.

Initially, we consider that N_b and N_r notated in algorithm 1 are 128-bit and 12 rounds, and they can be expended. The basic notations in the proposed ARIA masking algorithm are shown as

- Let $ek = (ek_0, ek_1, \ldots, ek_{15})$ be a 128-bit round key, where ek_i

is an 8-bit subblock of ek for $i = 0,\ldots,15$.

- Let $m = (m_0, m_1,\ldots,m_{15})$ be a 128-bit message, where m_i is an 8-bit subblock of m for $i = 0,\ldots,15$.
- Let $x = (x_0, x_1,\ldots,x_{15})$ be a 128-bit substitution layer input, where x_i is an 8-bit subblock of x for $i = 0,\ldots,15$ and denoted as $x=m\oplus ek$.
- Let $y = (y_0, y_1,\ldots,y_{15})$ be a 128-bit substitution layer output, where y_i is an 8-bit S-box output of y for $i = 0,\ldots,15$.
- Choose $r = (r_0, r_1,\ldots,r_{15})$ (resp. $r' = (r'_0, r'_1,\ldots,r'_{15})$) be a 128-bit input (resp. output) masking random, where r_i (resp. r'_i) is an 8-bit subblock of r (resp. r') for $i = 0,\ldots,15$.
- Choose 8-bit random values r_I and r'_I.
- An initial masked inversion table MI is calculated with initial masks r_I and r'_I as shown in algorithm 3.

For each S-box, the proposed masked S-box, named the adaptive masked S-box, is defined as

$$MS(x'_i) := K\cdot MI((x'_i\oplus r_I)\oplus r_i)\oplus(r'_i\oplus K\cdot r'_I\oplus u),$$
$$MS^{-1}(x'_i) := MI(K^{-1}\cdot(x'_i\oplus u)\oplus(K^{-1}\cdot r_i\oplus r_I))\oplus(r'_I\oplus r'_i),$$

for some i=0,…,15, where $K = A, u = a$ (resp. $K = D, u = b$) for S_1-box (resp. S_2-box), and the i-th masked S-box input $x'_i = x_i\oplus r_i$. The adaptive masked S-box can be represented as algorithm 4.

There are sixteen masked S-box calculations in a round, and each calculation uses a different random number r_i, as shown in Fig. 1. The sequence of masked S-boxes in a substitution layer is alternated every round. Since $MS(x'_i) = S(x_i)\oplus r'_i$ and $MS^{-1}(x'_i) = S^{-1}(x_i)\oplus r'_i$ by the above definition of the masked S-boxes, $y' = y\oplus r'$, where y' is an output of the masked substitution layer. For example, the first block message m_0 in odd round is calculated to $x'_0 = m_0\oplus ek_0\oplus r_0$ and inserted into the MS_1 box with random masks r_0, r'_0, r_I, and r'_I. The output of MS_1 box is computed as $y_0' = S_1(x_0)\oplus r_0'$.

```
Algorithm 3. Adaptive masked inversion table
    Input: Inversion table $I$, $r_I$, $r'_I$
    Output: Adaptive masked inversion table $MI$
    for $j$=0 to 255 do
        $MI[j]=I[j\oplus r_I]\oplus r'_I$
    end for
    return $MI$
```

```
Algorithm 4. Adaptive masked S-Box
    Input: $x'_i$, $r_i$, $r'_i$, $MI$, $r_I$, $r'_I$, affine table $A$, $D$, $A^{-1}$, $D^{-1}$, $a$, $b$, $case$
    Output: Masked S-box output $y'_i$
    $y'_i = \begin{cases} A[MI[(x'_i\oplus r_I)\oplus r_i]]\oplus r'_i\oplus A[r'_I]\oplus a, & case = MS_1, \\ D[MI[(x'_i\oplus r_I)\oplus r_i]]\oplus r'_i\oplus D[r'_I]\oplus b, & case = MS_2, \\ MI[A^{-1}[x'_i\oplus a]\oplus A^{-1}[r_i]\oplus r_I]\oplus r'_I\oplus r'_i, & case = MS_1^{-1}, \\ MI[D^{-1}[x'_i\oplus b]\oplus D^{-1}[r_i]\oplus r_I]\oplus r'_I\oplus r'_i, & case = MS_2^{-1}, \end{cases}$
    return $y'_i$
```

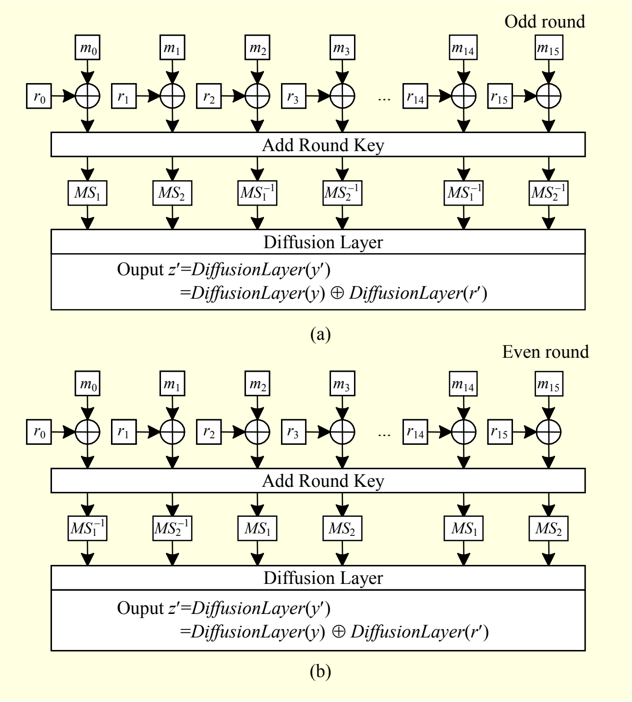


Fig. 1. Proposed ARIA masking scheme.

Because the proposed algorithm uses different mask values in a round, the diffusion layer output z' includes random masks as

$$z' = DiffusionLayer(y')$$
$$= DiffusionLayer(y)\oplus DiffusionLayer(r').$$

Therefore, we can consider the diffusion output with random numbers for the next round of operations. When the n-th round key is denoted as $ek(n)$ and the n-th round random mask r' is defined as $r'(n)$, the $(n+1)$th round key of the proposed masking algorithm is calculated as $DiffusionLayer(r'(n))\oplus ek(n+1)$.

IV. Proposed Masking Algorithm Evaluation with Software Implementation

The implementation results of the previous studies on masking algorithms are focused on software. Because there are no comparable reports on the hardware implementation of masked ARIA, we evaluate the proposed adaptive random masking algorithm with software implementation. We implement the proposed algorithm to software and compare it with previous algorithms within operation speed, memory allocation for algorithm, and security.

1. Algorithm Optimization and Software Implementation

In general, the masking scheme makes masking S-box tables, but our proposed implementation scheme does not make masking S-box tables. The proposed algorithm generates masked inversion table and calculates the output of S-boxes every round. Therefore, the proposed algorithm can be implemented with only 256 bytes of ROM and 256 bytes of RAM. However, this implementation proved too slow in calculating affine transforms. As such, we changed from calculating affine transforms to using ROM tables. Also, we can improve operation speed by approximately 90%. Thus, our implementation needs 1,024 bytes of ROM for the affine transform tables, 256 bytes of ROM for an inversion table, and 256 bytes of RAM for a masked inversion table.

The proposed masking procedure is composed of the following three steps which are shown in the previous section. First of all, the masked inversion table is made from an inversion table and random masks. Second, the masked substitution layer is calculated at a round with input values and random masks. In this case, management for random numbers is needed because our substitution layer can be made using different random values for every input byte in a round operation to protect from SODPA. Finally, the diffusion layer is calculated with random values. As the diffusion layer output includes random values, the round keys which are used in next round operation are considered with the random masks.

Our proposed algorithm is fully implemented in the following three types:

- The first and final round masking method is the fastest way to mask. The method changes random numbers for every block, and the method has just 4% overhead compared to the original ARIA algorithm in clock time. This method has security problems, but it is just an example for fast operation for comparison to other method.
- The second method is implemented with full round masks. The second method changes random masks for every block, but it uses the same random values which were used in the first round for all rounds. This method has approximately 11% overhead in clock time.
- Full round masking with different random values at every round method changes random masks in every block and every round. This method requires twice as much operation time as original ARIA, but this method offers high security.

All implementations are simulated by using CalmSHINE software, and the results are shown in Table 2.

2. Algorithm Comparison with Software Implementation

Generally, the ARIA masking algorithm needs 1,024 bytes of RAM and 1,024 bytes of ROM to generate masking tables. In contrast, our proposed algorithm can be implemented with 256 bytes of ROM and 256 bytes of RAM. However, for speed, we implement the proposed algorithm with 1,280 bytes of ROM and 256 bytes of RAM. As the affine computations are very slow, we changed from computing affine transform to affine table indexing. The proposal can be efficiently applied in resource limited environments like smart cards which have small RAM size.

Table 2. Clock cycles for calculation of proposed masking.

	ARIA	Masking [10]	Proposed ARIA masking		
Masking Option	-	Full round masking	1st, final round masking	Full round masking	Full round masking
Random changes	-	Every block	Every block	Every block	Every block and round
Calculation clocks (%)	250,668 (100%)	- (2,512%)	260,850 (104%)	279,104 (111%)	498,815 (199%)

A. Proposed Algorithm Evaluation with Previous Algorithm Using a Masked Inversion and Affine Transforms

The operation speeds of previous algorithms are presented in Table 1. From the survey, we can find that the previous algorithm composed with a masking inversion table and affine transforms is very slow in contrast with the proposed algorithm. As we do not implement the previous work [10], the comparison is preceded using the relative times for operation in Table 2. When we compare our proposal with the previous method, which uses different random masks for every block, the proposed masking method is extremely faster than the other shown in Table 2. Moreover, not only can our proposal change random values for every block but, in contrast with the previous method, can also use different masks for every round.

B. Proposed Algorithm Evaluation with Previous Algorithms Using Masked S-Box Tables

It is hard to correctly compare operation times of the proposed algorithm with other masking algorithms which generate masked S-box tables because our implementation does not make masking tables. In this case, we can compare our algorithm using two methods.

First, we count operations. The proposed algorithm requires 832 table indexing and 1,664 XOR operations to generate a masked inversion table and calculate masked S-boxes. As such the proposed algorithm needs less table indexing but more XOR operations compared with the algorithm from [7]. As the count result shown in Table 3 is ambiguous to exactly compare

Table 3. Cost comparison of generating masked S-boxes and round operation.

Resources and operations		General masking	Masking [7]	Proposed masking
Required resource	ROM (byte)	1,024	512	1,280
	RAM (byte)	1,024	1,024	256
Mask table generation	Table index	1,216	708	256
	XOR	2,048	768	512
SubLayer calculation	Table index	192	192	576
	XOR	0	0	1,152

Table 4. Operation clock cycle comparison by simulation.

Simulation clock cycle		General masking	Masking [7]	Proposed masking
CalmRISC	Table generation	35,097	25,881	11,544
	SubLayer/round	448	448	1,024
MSP430	Table generation	15,635	12,312	5,651
	SubLayer/round	128	128	352
ATmega128	Table generation	17,171	13,105	6,419
	SubLayer/round	304	304	640

the proposed algorithm and the others, we simulate the algorithms. When we use the CalmSHINE, which is a smart card software development toolkit to simulate the algorithms for comparison of operation clock cycles, 35,097 clocks are required for the generation of four masked S-boxes in general masking method. Our proposed implementation, however, needs 11,544 clock cycles for making a masked inversion table. Calculations for the substitution layer in a round require 1,024 clock cycles which is double the general masking algorithm and the algorithm in [7]'s calculation time.

We also simulate the implements using various development tools while considering a diverse range of embedded platforms. We organize the same algorithms for MSP430 and ATmega-128, which are widely used microcontrollers in various fields, and we can get similar result in terms of clock cycles. As a result, the proposed algorithm is secure against the SODPA and fast, even though the number of operation cycles for masking the substitution layer is greater than for other methods. The whole results are shown in Table 4.

V. Adaptive Random Masking H/W Implementation

Although there is much research on the ARIA hardware modules which do not include masking and the related architectures [6], [11]-[13], there are no studies on comparison of efficiency and structures for masked ARIA hardware implementation. In this section, we describe the masked hardware architectures considering previous architectures, implement the proposed algorithm to hardware modules, and compare the designs.

1. Proposed Masking Hardware Architecture

Generally, the ARIA hardware is implemented to 128-bit, 32-bit, 16-bit, or 8-bit architectures, but there are no reports of masked ARIA hardware implementation. Therefore, we considered three types of designs to compare area efficiency, performance, and power consumption themselves. The implementations are considered using different random masks for every block in every round.

- The fundamental method uses a 128-bit structure. The 128-bit architecture has a 128-bit diffusion layer composed of 480 two-input XOR gates and processes all random masks at once. Therefore, the whole structure is composed with a 128-bit substitution layer as shown in Fig. 2(a).
- The diffusion layer is initially a 16×16 matrix, but it can be split into four 16×4 matrices. Because the matrix C in session 2 is an involution matrix, elements of C are described as $c_{ij} = c_{15-i,8+j}$. Thus, the second architecture can be made using a 32-bit architecture with a 32-bit diffusion layer and a 32-bit masking process as shown in Fig. 2(b).
- The architecture as shown in Fig. 2(c) has mixed structure composed of an 8-bit substitution layer and a 16-bit diffusion layer. The substitution layer has just a masked S-box composed by a masked inversion and an affine transform. In the diffusion layer, we can denote the diffusion layer input $I_n = (i_{n,0}, i_{n,1}, \ldots, i_{n,7})$ where $\{n \mid 0 \le n \le 15\}$ because the element I_n of input vector I is an 8-bit value. Furthermore, the array of the m-th bit selected from each input byte is defined as $\hat{I}_m = (i_{0,m}, i_{1,m}, \ldots, i_{15,m})$, where $\{m \mid 0 \le m \le 7\}$. That is, the result value can be described as $\hat{O}_m = C \cdot \hat{I}_m$ where $\{m \mid 0 \le m \le 7\}$.

2. Optimization for the Substitution Layer

Making a small S-box is very important because an S-box is large and spends much power in cryptographic devices. As masked S-boxes are composed by a masked inversion function and affine transforms, a small masked inversion module is important for an area efficient implementation. Because a multiplication requires more circuits than any other operation, the whole circuit size is shrunk by reducing multiplications [7], [14]-[18].

A masked inversion function is a combination of multiplications, squares, constant multiplications, and XORs. Although an arbitrary inversion has fewer multiplications, the

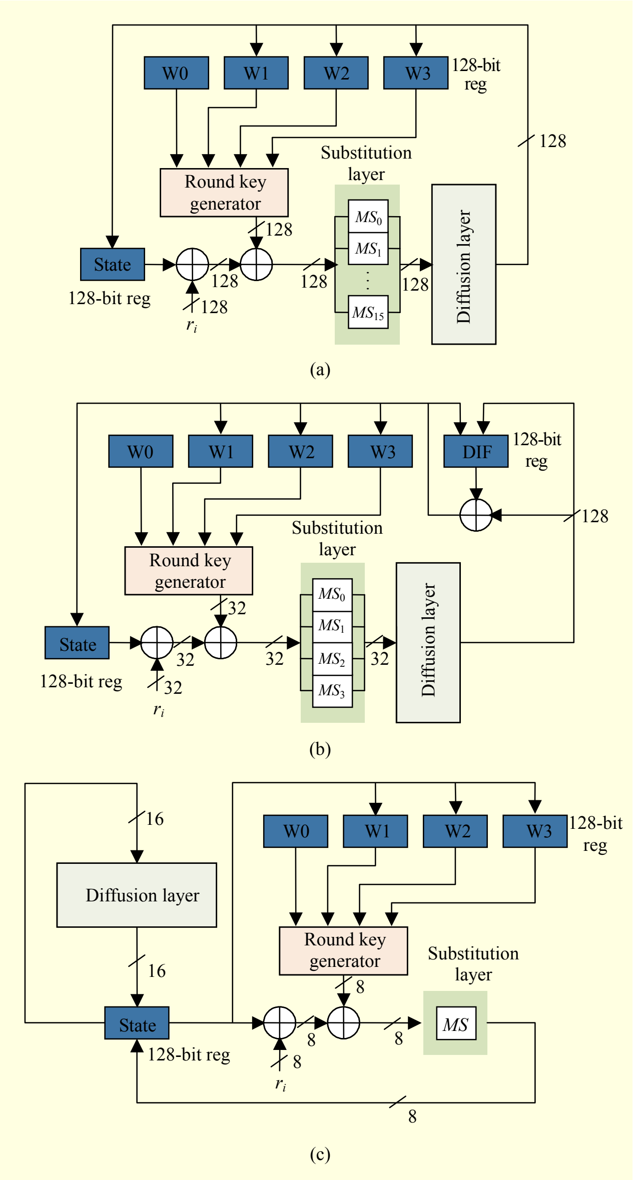


Fig. 2. Simple block diagram of hardware implementation for proposed masking algorithm: (a) 128-bit structure, (b) 32-bit structure, and (c) 8/16-bit structure.

whole module size is not always reduced. Moreover, the area efficiency can be changed by the base field. For example, the inversion of [7] is smaller than [17] over the composite field $GF(((2^2)^2)^2)$, but the inversion of [7] is larger than [17] in the field $GF((2^2)^2)$. These kinds of results are shown in Table 5 which includes the number of transistors to compare module size exactly, and the proposed adaptive inversion is optimized based on previous studies.

The proposed masked S-box, named the adaptive masked S-box, is implemented with an optimized adaptive masked inversion, and the architecture is accomplished with the six steps shown in Fig. 3.

Table 5. Comparison of previous masked inversion.

	Inversion [7]		Inversion [17]	
	$GF((2^2)^2)^2$	$GF(2^2)^2$	$GF((2^2)^2)^2$	$GF(2^2)^2$
Multiplies	7		8	
Squares	3		4	
Constant mult.	2		2	
XORs	23		18	
2-input AND	112	28	128	32
2-input XOR	205	72	200	66
Total transistors	3,132	1,032	3,168	984

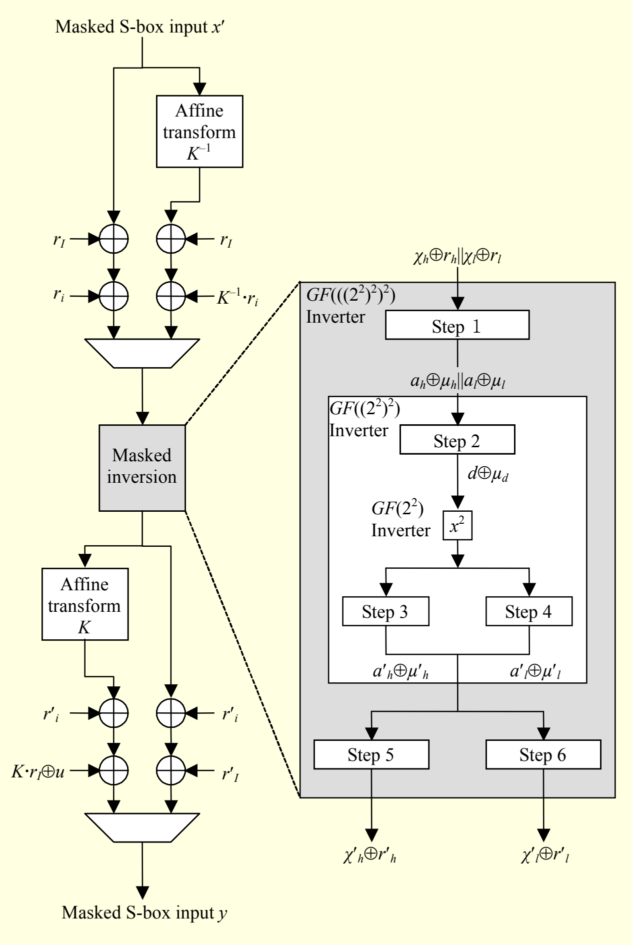


Fig. 3. Adaptive masked S-box structure.

When the 8-bit input of the adaptive masked inversion function is defined as $\chi \oplus r$ and the λ is the constant value, the first step output $\alpha \oplus \mu$ is calculated as

$$\textbf{Step 1: } \alpha \oplus \mu = (((r'_l \oplus M_A) \oplus M_C) \oplus ((\mu \oplus M_B) \oplus M_D)) \oplus ((r'_l \oplus \lambda M_F) \oplus M_E),$$

where

$$
\begin{aligned}
M_A &= \lambda(\chi_h \oplus r_h)^2, \\
M_B &= (\chi_l \oplus r_l)((\chi_h \oplus r_h) \oplus (\chi_l \oplus r_l)), \\
M_C &= r_h((\chi_h \oplus r_h) \oplus (\chi_l \oplus r_l)), \\
M_D &= r_h r_l, \\
M_E &= (\chi_h \oplus r_h)(r_h \oplus r_l) \oplus r_l^2, \\
M_F &= r_h^2.
\end{aligned}
$$

Steps 2 to 4 are calculated as

$$
\begin{aligned}
\textbf{Step 2: } d \oplus \mu_d = \rho(\alpha_h \oplus \mu_h)^2 &\oplus ((\alpha_h \oplus \mu_h)(\alpha_l \oplus \mu_l)) \\
&\oplus (\alpha_l \oplus \mu_l)^2 \oplus (\alpha_l \oplus \mu_l)\mu_h \\
&\oplus (\alpha_h \oplus \mu_h)\mu_l \oplus \rho\mu_h^2 \oplus \mu_l^2 \oplus \mu_h\mu_l \oplus \mu_h
\end{aligned}
$$

$$
\begin{aligned}
\textbf{Step 3}: \alpha'_h \oplus \mu'_h = (\alpha_h \oplus \mu_h)(d' \oplus \mu_l) &\oplus (d' \oplus \mu_l)\mu_h \\
&\oplus (\alpha_h \oplus \mu_h)\mu_l \oplus \mu_h\mu_l \oplus \mu_l
\end{aligned}
$$

$$
\begin{aligned}
\textbf{Step 4}: \alpha'_l \oplus \mu'_l = (\alpha'_h \oplus \mu'_h) &\oplus (\alpha_l \oplus \mu_l)(d' \oplus \mu_h) \\
&\oplus (d' \oplus \mu_h)\mu_l \oplus (\alpha_l \oplus \mu_l)\mu_h \\
&\oplus \mu_h \oplus \mu_l \oplus \mu_h\mu_l
\end{aligned}
$$

Finally, steps 5 and 6 are compute as

$$\textbf{Step 5: } \chi'_h \oplus r'_h = r'_h \oplus M_E \oplus M_G \oplus r'_l \oplus M_H$$

$$
\begin{aligned}
\textbf{Step 6: } \chi'_l \oplus r'_l = (((\chi_h \oplus r_h) &\oplus (\chi_l \oplus r_l) \\
&\oplus r_h)(\alpha' \oplus \mu') \oplus M_C) \oplus M_H,
\end{aligned}
$$

where

$$
\begin{aligned}
M_G &= ((\chi_h \oplus r_h) \oplus (r_h \oplus r_l))((\alpha' \oplus \mu') \oplus r_l), \\
M_H &= (\alpha' \oplus \mu')r_l \oplus r'_l \oplus M_F \oplus M_D.
\end{aligned}
$$

3. Implementation Results and Performance Evaluation

We implement three types of hardware using the proposed adaptive random masking algorithm. The adaptive masked inversion in the adaptive masked S-box requires approximately 4,116 transistors, and it can reduce about 1.2% of the whole size compared to the previous inversion architecture [7]. All designs are synthesized without random number generator modules by Synopsys with 0.25-µm CMOS standard-cell technology library, and the results are shown in Table 6.

When we compare our 128-bit masked ARIA module to non-masked ARIA [6], the proposed masking module has low throughput. However, the proposed architectures of 32-bit and 16-bit are more efficient within throughput as shown in Table 6. Our non-masked ARIA, which is composed with the proposed 8/16-bit structure, is relatively larger than a previous work [12]. That is, the previous work [12] has latches for memory. It is hard to reduce the size of ARIA modules because of memories in fact. The memories composed with latches can reduce about 20% of the whole size.

Table 6. Performance comparison of masked ARIA and non-masked ARIA hardware implementation.

	Architecture	Gate count	Freq. (MHz)	Thrt. (Mbps)	Thrt/area (kbps/gate)
ARIA	[6]	21,757	97	827	38.01
	[11]	13,893	71	25	1.79
	[12]	6,840	15	3.93	0.57
	Our 8/16-bit	10,027	47	15	1.59
Proposed masked ARIA	128-bit	29,638	35	298	10.08
	32-bit	17,484	30	60	3.43
	8/16-bit	10,740	38	12	1.20

Table 7. Comparison of power simulation result.

(µW)

Proposed masked ARIA			ARIA	AES [19]
128-bit	32-bit	8/16-bit	8/16-bit	
118.95	81.54	47.47	34.96	20.38

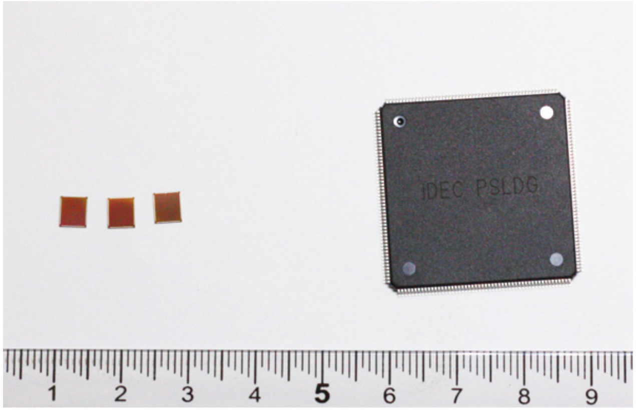


Fig. 4. Proposed masked ARIA implementation in ASIC chips.

We performed a power simulation of our designs at 100 kHz 2.5 V using the synopsys power compiler as shown in Table 7. Our implementation has random number generators to make masks, but the simulation was run without the random number generator. Finally, we produced our design to ASIC chips using 0.18-µm CMOS standard-cell technology library as shown in Fig. 4 and the security evaluation board for the ASIC chip as shown in Fig. 5.

It is impossible to compare our outcomes accurately with previous works because there are no reports that compare area efficiency or the performance of masked ARIA hardware implementations. Therefore, we implement the proposed algorithm on various architectures and compare them. The result shows that the 8/16-bit architecture is small and has low power consumption, which makes it suitable for small devices

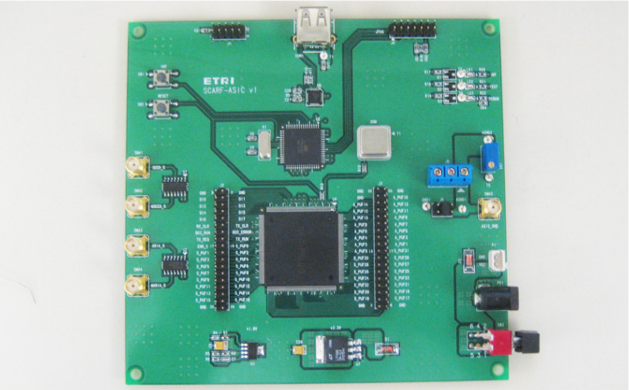


Fig. 5. Test board for proposed masked ARIA implementation.

like RFIDs or smart cards. However, our module is large and uses a great deal of power compared with the AES architecture [19], but our module is not large considering that the original ARIA architecture is basically larger than the AES. When the non-masked ARIA module is implemented using the proposed 8/16-bit architecture, the equivalent gate count is 10,027 and the average power consumption at 100 kHz 2.5 V is 34.96 µW. Therefore, we consider that the overhead of our masking scheme has a 7.1% increment in area and 35.8% increment in power consumption. Thus, compared with other ARIA modules, not only is our design efficient but also safe against SODPA.

VI. Security Analysis

1. Theoretical Security Proof

When an arbitrary element x and a uniformly distributed element r which is independent of x in $GF(2^n)$ are given, the result of the addition and the result of multiplication between x and r are independent of x over the composite field $GF(2^n)$ [7], [17]. Because our masking algorithm follows this theory, we can easily prove security of the proposed algorithm as seen in previous studies [7], [17].

2. Experimental Security Proof

We implemented our masking algorithm in software and hardware, and then experimentally verified the security for our system. We performed the SODPA resistant test using a second-order correlation power analysis (SOCPA) [3], [20].

i) We implemented the general masking algorithm on an ATmega128 processor, and then performed the SOCPA. The result shown in Fig. 6 presents that the general masking method has security problems against the SOCPA attack.

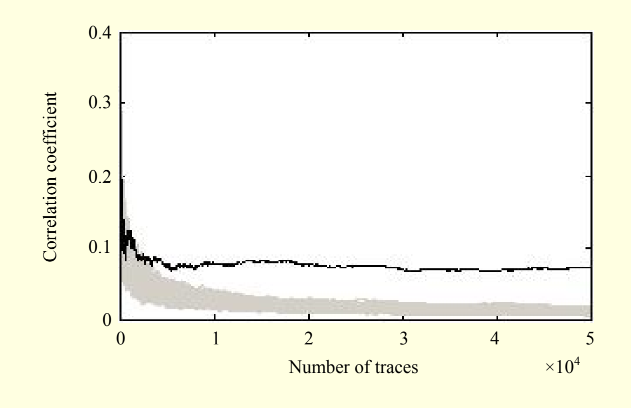


Fig. 6. SOCPA for various numbers of traces in software implementation of general masking.

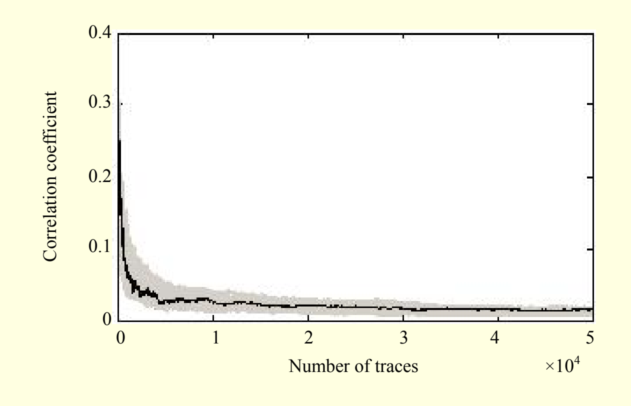


Fig. 7. SOCPA for various numbers of traces in software implementation of proposed masking.

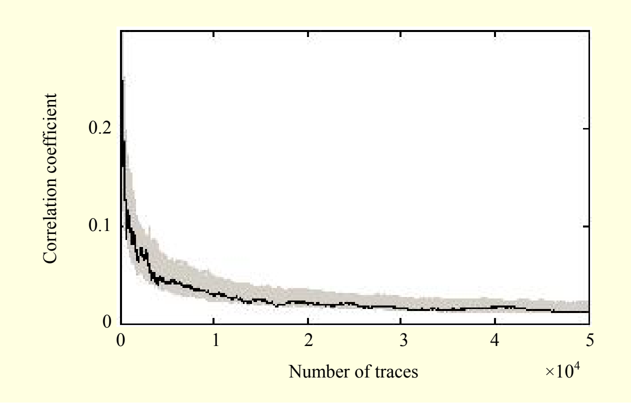


Fig. 8. SOCPA for various numbers of traces in ASIC of proposed masking.

ii) We implemented the adaptive random masking algorithm on the same ATmega128 processor, performed a SOCPA experiment with the same conditions of the first test. The proposed masking algorithm is secure against SOCPA, and a result of the experiment is shown in Fig. 7.

iii) We performed the same experiment on an ASIC chip which includes the adaptive random masking algorithm. We made an evaluation board as shown in Fig. 5 and performed SOCPA on the board. From the experiment, we can verify that the masked ARIA hardware module is secure against SOCPA, and the result is shown in Fig. 8.

VII. Conclusion

This article presented the secure hardware implementation and an advanced masking algorithm for block cipher ARIA. The proposed algorithm, the adaptive random masking algorithm, was verified using comparison results of software implementations, and we implemented it in hardware. As this paper is the first report of masked ARIA hardware implementation, we compared various hardware architectures. The proposed 8/16-bit module, which is the smallest of our work, uses only 47.47 µW at 100 kHz 2.5 V, and the overhead of the masking scheme is only a 7.1% increment in area and a 35.8% increment in power consumption compared to non-masked ARIA module. Finally, we experimentally verified the security of the proposed algorithm, and results showed the proposed algorithm and implementations were secure against SODPA. Thus, we conclude that the proposed hardware and the adaptive random masking algorithm are appropriate for small devices like RFIDs or smart cards.

References

[1] P. Kocher, J. Jaffe, and B. Jun, “Differential Power Analysis,” *Proc. CRYPTO*, *LNCS*, vol. 1666, 1999, pp. 388-397.

[2] S. Mangard, E. Oswald, and T. Popp, *Power Analysis Attacks: Revealing the Secrets of Smart Cards*, Springer, 2007.

[3] T. Messerges, “Using Second-Order Power Analysis to Attack DPA Resistance Software,” *Proc. CHES*, *LNCS*, vol. 1965, 2000, pp. 238-251.

[4] National Security Research Institute: *The ARIA Specification*, http://210.104.33.10/ARIA/index-e.html

[5] D. Kwon et al., “New Block Cipher: ARIA,” *Proc. ICISC*, *LNCS*, vol. 2971, Nov. 2003, pp. 432-445.

[6] S. Lee, S. Moon, and J. Kim, “High-Speed Hardware Architectures for ARIA with Composite Field Arithmetic and Area-Throughput Trade-Offs,” *ETRI J.*, vol. 30, no. 5, Oct. 2008, 2008, pp. 696-706.

[7] H. Kim et al., “Efficient Masking Method Appropriate for the Block Ciphers ARIA and AES,” *ETRI J.*, vol. 32, no. 3, June 2010, pp. 370-379.

[8] C. Kim, M. Schläfferm, and S. Moon, “Differential Side Channel Analysis Attack on FPGA Implementations of ARIA,” *ETRI J.*, vol. 30, no. 2, Apr. 2008, pp. 315-325.

[9] W. Li, D. Gu, and J. Li, “Differential Fault Analysis on the ARIA Algorithm,” *Info. Sci.*, vol. 178, no. 19, 2008, pp. 3727-3737.

[10] H. Yoo et al., “A Secure Masking-Based ARIA Countermeasure for Low Memory Environment Resistant to Differential Power Attack,” *J. KIISC*, vol. 16, 2006, pp. 143-155.

[11] J. Park et al., “Low Power Compact Design of ARIA Block Cipher,” *Proc. ISCAS*, IEEE, 2006, pp. 313-316.

[12] S. Yang, J. Park, and Y. You, “The Smallest ARIA Module with 16-Bit Architecture,” *Proc. ICISC*, *LNCS*, vol. 4296, 2006, pp. 107-117.

[13] 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. 820-822.

[14] A. Satoh et al., “A Compact Rijndael Hardware Architecture with S-Box Optimization,” *Proc. ASIACRYPTO*, *LNCS*, vol. 2248, 2001, pp. 239-254.

[15] J. Wolkerstorfer, E. Oswald, and M. Lamberger, “An ASIC Implementation of the AES S-Boxes,” *Proc. CT-RSA*, *LNCS*, vol. 2271, 2002, pp. 67-78.

[16] P. Chodowiec and K. Gaj, “Very Compact FPGA Implementation of the AES Algorithm,” *Proc. CHES*, *LNCS*, vol. 2779, 2003, pp. 319-333.

[17] E. Oswald et al., “A Side-Channel Analysis Resistant Description of the AES S-Box,” *Proc. FSE*, *LNCS*, vol. 3557, 2005, pp. 413-423.

[18] B. Zakeri et al., “Compact and Secure Design of Masked AES S-Box,” *LNCS*, vol. 4861, 2007, pp. 216-229.

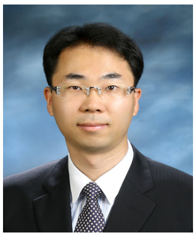
[19] M. Feldhofer, S. Dominikus, and J. Wolkerstofer, “Strong Authentication for RFID System Using the AES Algorithm,” *Proc. CHES*, *LNCS*, vol. 3156, 2004, pp. 357-370.

[20] E. Brier, C. Clavier, and F. Olivier, “Correlation Power Analysis with a Leakage Model,” *Proc. CHES*, *LNCS*, vol. 3156, 2004, pp. 135-152.

Junki Kang received his BS in electronics engineering from Chungnam National University, Daejeon, Rep. of Korea, in 2007. Since 2008, he has been a member of ETRI, Daejeon, Rep. of Korea, and pursuing the MS/PhD integrative program at the University of Science and Technology, Daejeon, Rep. of Korea. His main research interests include VLSI design, embedded system security, side-channel analysis, and information security.

Dooho Choi received his BS in mathematics from Sungkyunkwan University, Seoul, Rep. of Korea, in 1994, and the MS and PhD in mathematics from Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Rep. of Korea, in 1996 and 2002, respectively. Since January 2002, he has been a senior researcher at ETRI, Daejeon, Rep. of Korea. His current research interests are side channel analysis and its resistant crypto design, security technologies of RFID and wireless sensor network, lightweight cryptographic protocol/module design, and cryptography based on non-commutativity. He was an editor of the ITU-T Rec. X.1171.

Yong-Je Choi received his BSEE and MS from Chonnam National University, Kwangju, Rep. of Korea, in 1996 and 1999, respectively. He is currently a senior member of technical staff at ETRI, Daejeon, Rep. of Korea. His research interests include VLSI design, crypto processor design, side channel analysis, and information security.



Dong-Guk Han received his BS and MS in mathematics from Korea University, Seoul, Rep. of Korea, in 1999 and 2002, respectively. He received his PhD in engineering in information security from Korea University in 2005. He was a postdoctoral researcher in Future University-Hakodate, Japan. After finishing the doctoral course, he was an exchange student in the Department of Computer Science and Communication Engineering, Kyushu University, Japan, from April 2004 to March 2005. He was a senior researcher in ETRI, Daejeon, Rep. of Korea, from June 2006 to February 2009. He is currently working as an assistant professor with the Department of Mathematics, Kookmin University, Seoul, Rep. of Korea. He is a member of KIISC, IEEK, and IACR.