Efficient Masked Implementation for SEED Based on Combined Masking

HeeSeok Kim, Young In Cho, Dooho Choi, Dong-Guk Han, and Seokhie Hong

This paper proposes an efficient masking method for the block cipher SEED that is standardized in Korea. The nonlinear parts of SEED consist of two S-boxes and modular additions. However, the masked version of these nonlinear parts requires excessive RAM usage and a large number of operations. Protecting SEED by the general masking method requires 512 bytes of RAM corresponding to masked S-boxes and a large number of operations corresponding to the masked addition. This paper proposes a new-style masked S-box which can reduce the amount of operations of the masking addition process as well as the RAM usage. The proposed masked SEED, equipped with the new-style masked S-box, reduces the RAM requirements to 288 bytes, and it also reduces the processing time by 38% compared with the masked SEED using the general masked S-box. The proposed method also applies to other block ciphers with the same nonlinear operations.

Keywords: Side channel attacks, countermeasure, masking method, SEED.

Manuscript received Mar. 15, 2010; revised Sept. 13, 2010; accepted Oct. 4, 2010.

This research was supported by the Korea University Grant, and by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (20100024870), and also by the SCARF project which is the R&D program of KCA [Development of the Technology of Side Channel Attack Countermeasure Primitives and Security Validation].

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

Dooho Choi (email: dhchoi@etri.re.kr) is with the Software Research Laboratory, 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.11.1510.0112

I. Introduction

During the past few years, much research on differential power analysis (DPA) [1] 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 first-order DPA (FODPA) attacks [2]-[8].

In a masked implementation, all intermediate values are concealed by a random mask. 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 $y' \oplus m'$ in order to obtain the ciphertext y corresponding to the message x (the way this masking is applied depends on the cryptographic algorithm). This prevents FODPA attacks because the random mask values make it impossible for an attacker to predict the intermediate values.

In the masking method, we must know m' to obtain the ciphertext y. Due to the nonlinear parts, the block cipher has m' values that vary according to the different x values. Determining the value m_x (the m' value corresponding to x) without exposing the intermediate values requires a large number of operations. When designing a masking method, we must consider preferentially the nonlinear parts of the block cipher.

1. Motivation

The nonlinear operations in the block cipher SEED [9] are the S-box operations and addition modulo 2^{32}. Thus, we must consider preferentially these nonlinear parts to construct the masking method. However, protecting SEED using the general masking method has the following two problems.

A. RAM Requirement of 512 Bytes for Masked S-Boxes

The S-box operation, which is well known as the nonlinear part of ARIA [10] and AES [11] as well as SEED, outputs $S(a) \oplus m_a$ from the input value $a \oplus m$. As the m_a value is nonlinear in a, we need to make this output masking value the same for each a value. To fulfill this function, the masked block cipher generally uses the masked S-box (MS) table [2]-[5]. The MS table is generated by computing $MS(x \oplus m) = S(x) \oplus m'$ with new random numbers m, m' before encryption or decryption. Here, the reason why the different masks m and m' are used is because if the two mask values are the same, they are not secure in devices with power consumption models such as the Hamming distance model. In this model, the power consumption is proportional to the Hamming weight of the difference $((x \oplus m) \oplus (S(x) \oplus m') = x \oplus S(x)$ where $m = m')$ of two successive data values.

This generation method for the MS requires one additional MS table for each S-box. This construction of the MS table requires 256 bytes of RAM for a 256-byte S-box. Whereas ROM is inexpensive for general embedded processors, RAM is expensive. Therefore, the use of the MS tables in SEED having two S-boxes, S_1 and S_2, may impose a heavy burden on some devices.

B. Excessive $Secure_{AtoB}$ Function Calls

To mask the addition operation modulo 2^{32}, which is another nonlinear part of SEED, the designer must safely construct the routine that outputs 32-bit masked value $(x+y) \oplus m_3$ from two 32-bit masked inputs $x \oplus m_1$ and $y \oplus m_2$. In the general construction to fulfill this function, conversion methods between Boolean masking and arithmetic masking are used for a secure addition operation. Whereas the conversion $Secure_{BtoA}$ from Boolean masking to arithmetic masking is very efficient [12], the conversion $Secure_{AtoB}$ from arithmetic masking to Boolean masking requires a number of operations, even though pre-computed tables are used [13], [14]. In the masked implementation of SEED, most computational time is actually consumed by this $Secure_{AtoB}$ function. Also, using a general type of MS table after the operation of masked addition in SEED requires one $Secure_{AtoB}$ function call per masked addition.

2. Our Contribution

As each round of SEED uses two S-boxes and three modular additions, a masked implementation of SEED requires 512 bytes of RAM corresponding to MS tables and 48 $Secure_{AtoB}$ function calls for 48 addition operations.

This paper proposes a new generation scheme for the MS table. The MS table in the proposed scheme outputs the Boolean masked value from the Arithmetic masked input value. This method reduces the number of $Secure_{AtoB}$ function calls from 48 to 16. Surely, because the proposed method is different from existing generation schemes, additional operations such as carry corrections are inevitably required. However, to solve this problem, we propose a method that uses only 32 bytes of RAM and a few additional operations.

We also develop a new equation that can reduce the RAM usage for the MS tables, MS_1 and MS_2, from 512 bytes to 256 bytes, where MS_1 and MS_2 are the MS tables of S_1 and S_2, respectively. The proposed method generates only the MS_2 table before en/decryption, and subsequently, the output value corresponding to MS_1 is computed using the MS_2 table and a few operations related to the proposed equation.

The 8-bit masked implementation of SEED using the new-style MS table reduces the processing time by 38% compared with the masked SEED using the general MS table. In addition, the size for all tables using RAM, including the MS_2 table and carry correction tables, is only 288 bytes.

The remainder of this paper proceeds as follows. Section II explains a block cipher SEED and the masked SEED constructed by the general masking method. Section III describes our masked implementation of SEED, which uses the new-style MS. Sections IV and V demonstrate the performance and security of the proposed method. Section VI concludes the paper.

II. Block Cipher SEED and General Construction of Masked SEED

This section presents the block cipher SEED and the general masked SEED. We first define the following symbols.

$\oplus$: the exclusive-OR

$\&$: the logical AND

$0x\ldots$: the hexadecimal representation

$l(x)$: the number of bytes of x

$u||v = u \cdot 2^8 + v \ (u, v \in GF(2^8))$

$x = x_{l(x)-1}||x_{l(x)-2}||\ldots||x_1||x_0 = \sum_{i=0}^{l(x)-1} x_i 2^{8i} \ (x_i \in GF(2^8))$

$(k)^4 = k||k||k||k \ (k \in GF(2^8))$

$x +_k y$: $x + y \bmod 2^k$ where $x, y \in GF(2^k)$

$x -_k y$: $x - y \bmod 2^k$ where $x, y \in GF(2^k)$

$x (+)_{32} y$ where $l(x) = l(y) = 4$: $\sum_{i=0}^{3} (x_i +_8 y_i) 2^{8i}$

$\lfloor u \rfloor$: the largest integer less than or equal to u

$x^{\gg t}$: $\left\lfloor \frac{x}{2^t} \right\rfloor$

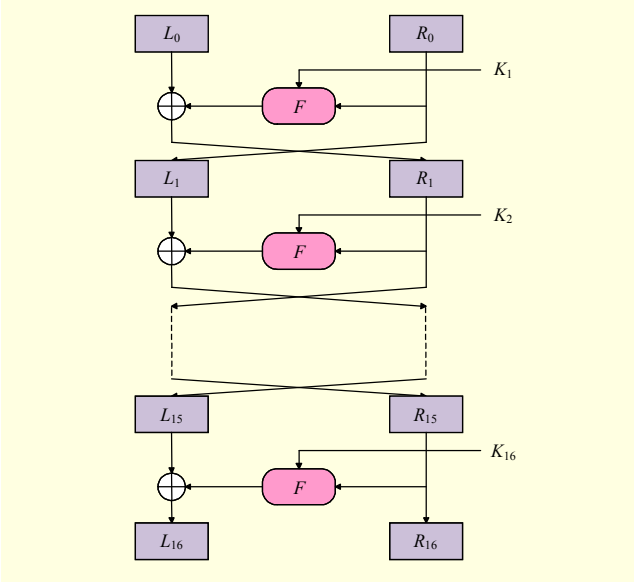


Fig. 1. Structure of SEED.

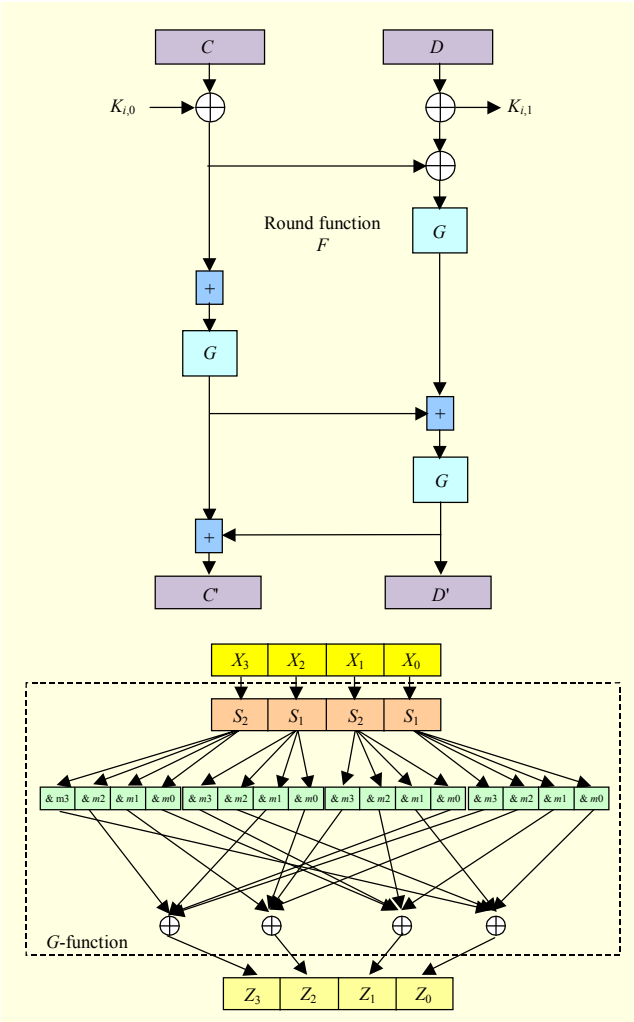


Fig. 2. Round function F and internal function G.

SEED is a block cipher designed in 1998 by the Korea Internet & Security Agency; it has been the national standard since 2000.

As shown in Fig. 1, SEED is a kind of classic Feistel structure cipher with 16 rounds. The 128-bit input of SEED is divided into two 64-bit blocks, and the second 64-bit block is the input to the round function F with a 64-bit round key generated from the key scheduling. Then, the output of the function F is XORed with the first 64-bit block.

The round function F divides the 64-bit input block into two 32-bit blocks (C and D) and carries out a mixing phase with two 32-bit round key blocks ($K_{i,0}$ and $K_{i,1}$). Next, three phases of the function G are computed with additions for mixing two 32-bit blocks. The round function F and the internal function G are depicted in Fig. 2.

In Fig. 2, the function G uses two S-boxes, S_1 and S_2. Each S-box is defined by the following equations.

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

$$S_1(x) = A_1 x^{247} \oplus a,$$

$$S_2(x) = A_2 x^{251} \oplus b,$$

where A_1 and A_2 are 8×8 matrices, and a (0x*a*9) and b (0x38) are 8×1 vectors.

Constructing the masked SEED as mentioned in the introduction requires two MS tables MS_1 and MS_2 satisfying the following equations, where m and m' are the 8-bit random numbers generated before encryption [2]-[5]:

$$MS_1(x \oplus m) = S_1(x) \oplus m',$$

$$MS_2(x \oplus m) = S_2(x) \oplus m'.$$

The general masked addition (MA) operation modulo 2^{32}, which is another nonlinear part of SEED, is constructed as follows, where $m''=m \oplus m'$, $m_1=m_2=m''||m''||m''||m''$, and $m_3=m||m||m||m$ [12]-[14].

Algorithm 1. General MA algorithm.
Input: 32-bit x' ($=x \oplus m_1$), y' ($=y \oplus m_2$), m_1, m_2, m_3.
Output: 32-bit $(x+y) \oplus m_3$.
1. $(x+m_1) = Secure_{\text{BtoA}}(x', m_1)$,
2. $(y+m_2) = Secure_{\text{BtoA}}(y', m_2)$,
3. $((x+y)+m_3) = ((m_3 + (x+m_1)) + (y+m_2)) - m_1 - m_2$,
4. Return $(x+y) \oplus m_3 = Secure_{\text{AtoB}}((x+y)+m_3, m_3)$.

The construction of the general masked function F using the above two MS tables and MA operations is depicted in Fig. 3.

This construction of masked SEED requires 512 bytes of RAM corresponding to the two MS tables and 48 (=3×16 round) $Secure_{\text{AtoB}}$ function calls. Thus, the SEED becomes slow and consumes excessive amounts of RAM. Therefore, this construction imposes a heavy burden on embedded devices.

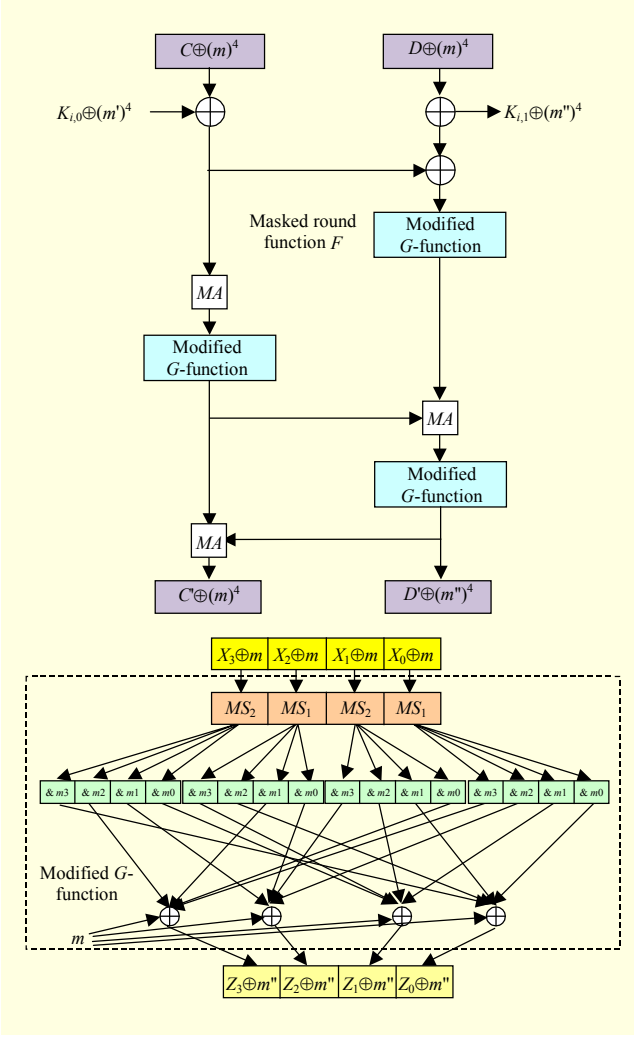


Fig. 3. General construction of masked function F.

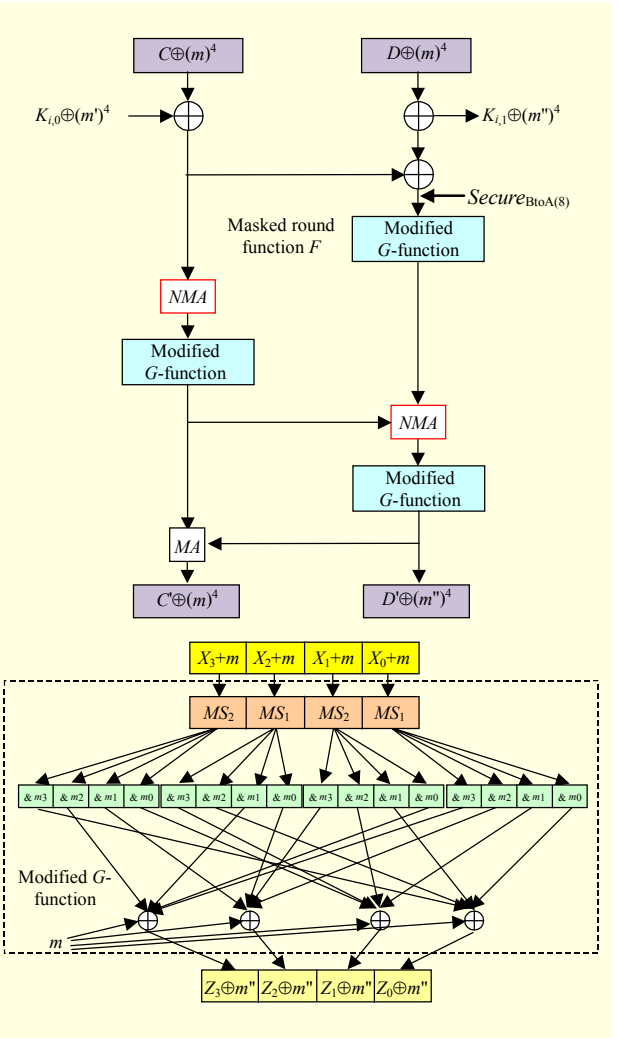


Fig. 4. Construction of masked function F using new-style MS.

III. Efficient Masked Implementation of SEED Using New-Style Masked S-Box

1. New-Style Masked S-Box

This subsection proposes a new-style MS that can reduce the number of $Secure_{AtoB}$ function calls. Our MSs are designed to satisfy the following equations, where m and m' are the 8-bit random numbers generated before encryption.

$$MS_1(x) = S_1(x \lll_8 m) \oplus m',$$

$$MS_2(x) = S_2(x \lll_8 m) \oplus m'.$$

We will explain a method that can compute the output values of two MSs using only 256 bytes of RAM in III.3 below. When designing MS tables such as the above equations, the first two MA operations in each round are replaced with the following new masked algorithm (NMA), where $m'' = m \oplus m'$, $m_1 = m_2 = m'' || m'' || m'' || m''$, and $m_3 = m || m || m || m$.

Algorithm 2. NMA.
Input: 32-bit x' $(= x \oplus m_1)$, y' $(= y \oplus m_2)$, m_1, m_2, m_3.
Output: 32-bit $(x +_{32} y) (+)_{32} m_3$.
1. $(x +_{32} m_1) = Secure_{BtoA}(x', m_1)$,
2. $(y +_{32} m_2) = Secure_{BtoA}(y', m_2)$,
3. $((x +_{32} y) +_{32} m_3) = ((m_3 +_{32} (x +_{32} m_1)) +_{32} (y +_{32} m_2)) -_{32} m_1 -_{32} m_2$,
4. Return $(x +_{32} y) (+)_{32} m_3 = CarryCorrection((x +_{32} y) +_{32} m_3, m_3)$.

Note that changing the form of the MS replaces the $Secure_{AtoB}$ function call with the *CarryCorrection* function call. However, the *CarryCorrection* function, which is presented in the next subsection, requires considerably fewer operations than the $Secure_{AtoB}$ function. The *CarryCorrection* function changes the 32-bit arithmetic masked value $a +_{32} t$ into 8-bit arithmetic masked values a $(+)_{32}$ t, where $l(a) = l(t) = 4$. Therefore, the output of the *NMA* operation is suitable as input

value of the MS tables in the modified function G. By this new-style MS, the construction of the masked function F in Fig. 3 is replaced as in Fig. 4.

2. Carry Correction Function: *CarryCorrection*

As previously mentioned, the 32-bit value before calling the *CarryCorrection* function in the first two NMA operations of each round is the 32-bit arithmetic masked value. We define this masked value as $x'=x_3'||x_2'||x_1'||x_0'$ $(=x_3||x_2||x_1||x_0 +_{32} m_3, m_3=m||m||m||m)$, where $x_3||x_2||x_1||x_0$ is the unmasked value. However, this value must be transformed into 8-bit arithmetic masked values $x_3||x_2||x_1||x_0$ $(+)_{32}$ m_3 to look up or compute the output values of the MS table. Therefore, the carry values generated in each byte of the 32-bit addition must be securely eliminated. To eliminate these carry values, we need to determine whether a carry is or is not generated in $x_3'||x_2'||x_1'||x_0'$ $(x_i'=x_i+m+carry(x_{i-1}') \bmod 2^8, carry(x_{-1}')=0)$ from the least significant byte to the most significant byte. To make this determination, we need to compare $x_i'-carry(x_{i-1}')$ with m securely. However, a comparison using '<' or '>' exposes the unmasked intermediate value x_i $(= (x_i'-carry(x_{i-1}')) - m)$. Thus, we propose a secure computation method of the carry values by using two 16 bytes carry tables generated before en/decryption. The reason behind making two 16-byte carry tables, C_1 and C_2, is that the general carry table comparing t $(=x_i'-carry(x_{i-1}'))$ with the random number m requires 256 bytes of RAM. The idea is straightforward. The first carry table, C_1, compares the most significant four bits of t and m. In addition, because the most significant four bits of t and m may be the same, C_2, a second carry table, is required to compare the least significant four bits of t and m. The following algorithm generates the two carry tables C_1 and C_2.

Algorithm 3. Generation scheme for carry tables C_1 and C_2.
Input: 8-bit random integers m, λ.
Output: 16 bytes carry tables C_1 and C_2.
1. For t=0x0 to 0xf do
1.1. If $t<(m\&0xf0)^{\gg 4}$, $C_1[t] = \lambda+1$;
1.2. Else if $t=(m\&0xf0)^{\gg 4}$, $C_1[t] = \lambda+2$;
1.3. Else, $C_1[t] = \lambda$;
1.4. If $t<(m\&0xf)$, $C_2[t] = \lambda+1$;
1.5. Else, $C_2[t] = \lambda$.
2. Return C_1 and C_2.

In the above algorithm, we use carry tables that are randomized by the 8-bit integer λ to prevent that the computed carry value would leak some information about x_i [13]. In the first two NMA operations in each round, the *CarryCorrection* function performs the following algorithm by using λ and the carry tables, C_1 and C_2, generated before en/decryption.

Algorithm 4. *CarryCorrection* function.
Input: $x'=x_3'||x_2'||x_1'||x_0'$ $(=x_3||x_2||x_1||x_0 +_{32} m_3, m_3=m||m||m||m)$.
Output: $y'=y_3'||y_2'||y_1'||y_0'$ $(= y_3, y_2, y_1, y_0 (+)_{32} m_3)$.
1. For i=0 to 2 do
1.1. $carry_1 = C_1[(x_i\&0xf0)^{\gg 4}]$;
1.2. $carry_2 = C_2[(x_i\&0xf)]$;
1.3. $carry = (carry_1 == \lambda+2)$? $carry_2$: $carry_1$;
1.4. $x_3'||...||x_{i+1}' = x_3'||...||x_{i+1}' -_{8(3-i)} carry$;
1.5. $x_3'||...||x_{i+1}' = x_3'||...||x_{i+1}' +_{8(3-i)} \lambda$.
2. Return $y = x_3'||x_2'||x_1'||x_0'$.

3. New Equation of Masked S-Box to Reduce RAM Size

As previously mentioned, the general masked SEED requires 512 bytes of RAM for MSs, but the proposed method uses only 256 bytes of RAM. To reduce RAM usage, we generate only the MS_2 table. Then, the output value corresponding to the MS_1 table is computed from the output value of the MS_2 table by a new equation between MS_1 and MS_2.

We first show the equation between S_1 and S_2 as follows, where B is an 8×8 matrix such that $B(x)=x^2$:

$$S_1(x) = A_1 x^{247} \oplus a = A_1 x^{-8} \oplus a ,$$

$$S_2(x) = A_2 x^{251} \oplus b = A_2 x^{-4} \oplus b ,$$

$$\begin{aligned} S_1(x) &= A_1((x^{-4})^2) \oplus a \\ &= A_1\left((A_2^{-1}(S_2(x) \oplus b))^2\right) \oplus a \\ &= A_1\left(B(A_2^{-1}(S_2(x) \oplus b))\right) \oplus a \\ &= (A_1 B A_2^{-1}) S_2(x) \oplus (A_1 B A_2^{-1} b \oplus a) \\ &= A_3 S_2(x) \oplus c. \end{aligned}$$

From the above equation, we can find the equation computing the output value of the MS_1 table based on the MS_2 table.

$$MS_1(x) = S_1(x -_8 m) \oplus m' ,$$

$$MS_2(x) = S_2(x -_8 m) \oplus m' ,$$

$$S_1(x -_8 m) = A_3 S_2(x -_8 m) \oplus c ,$$

$$S_1(x -_8 m) \oplus m' = A_3 S_2(x -_8 m) \oplus c \oplus m' ,$$

$$\begin{aligned} MS_1(x) &= A_3\left(MS_2(x) \oplus m'\right) \oplus c \oplus m' \\ &= A_3 MS_2(x) \oplus A_3 m' \oplus m' \oplus c \\ &= A_3 MS_2(x) \oplus c \oplus m_a \\ &= affine[MS_2(x)] \oplus m_a , \end{aligned}$$

where m_a is $A_3 m' \oplus m'$.

Table 1. Cost of generating MS tables in SEED.

	General method [2], [3], [5]-[8]	Proposed method
Clock cycles	8,935 cc	4,600 cc
RAM	512 bytes	256 bytes +32 bytes (carry table)
ROM	512 bytes	256 bytes

Table 2. Number of clock cycles of masked encryption in SEED.

SEED (5,962 cc)	General method [12], [13]	Proposed method
Clock cycles	118,480 cc	72,608 cc

Namely, if an affine table corresponding to $affine[t]=A_3t\oplus c$ is in ROM, the output value corresponding to $MS_1(x)$ can be obtained with only one table look-up and one-byte XOR operation.

A similar idea was published in [15]. However, our idea reduces RAM usage rather than ROM usage because we exploit the relation between two S-boxes rather than the relation between an S-box and its inverse; RAM is typically much more expensive than ROM.

IV. Performance Analysis

In short, the new-style MS has three advantages. First, we generate only one MS table before each SEED en/decryption. Therefore, we can reduce the time required to generate the MS table. Second, we can also reduce the RAM size because the proposed method does not generate the MS_1 table. Compared with the general method for generating MS tables [2], [3], [5]-[8], the proposed method reduces the required clock cycles and memory usage for the 8-bit implementation (Table 1). This numerical value was measured in an ATmega128 [16].

The third advantage of the new-style MS is the reduction of the time required to carry out the masked en/decryption scheme. This is because our MS can reduce the number of $Secure_{AtoB}$ function calls. We estimated the required clock cycles when constructing the $Secure_{AtoB}$ function using the technique described in [13]. As shown in Table 2, the required the number of clock cycles for an ATmega128 was lower in the proposed method than in the general method for masked encryption [12], [13].

Table 3 represents the clock cycles of masked SEED in an ATmega128, including MS table generation, the key schedule, and masked encryption.

Table 3. Total number of clock cycles for masked SEED.

SEED (12,302 cc)	General method	Proposed method
Clock cycles	134,950 cc	84,743 cc

V. Security Analysis

The security of the proposed masking method can be proven with lemmata 1, 2, and 3. As the proofs of lemmata 1 and 2 are straightforward, we omit them.

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

Two lemmata describe the basic reasons why arithmetic and Boolean masking methods are secure. Namely, if all intermediate values that can be attacked are concealed by random masks that are independent of these values, it is impossible for an attacker to predict the intermediate values. Therefore, the use of the two types of masking with the mask uniformly distributed completely prevents FODPA attacks.

We must consider the security of the equations of the MS, presented in III.3, because we do not confirm whether the intermediate values during this operation are concealed by the random integer uniformly distributed in $GF(2^n)$ or not. This security can be proven with the following lemma.

Lemma 3. The new equation of the MS in III.3 is secure.

Proof. The equation to compute the output value of $MS_1(x)$ is computed with the following three steps.

$a = MS_2(x) = S_2(x)\oplus m'$.
$b = Affine[a] = A_3a\oplus c = A_3S_2(x)\oplus c\oplus A_3m'$.
Return $b\oplus m_a = S_1(x)\oplus m'$.

The intermediate values in steps 1 and 3 are concealed by the uniform random number m'. Also, in step 2, the mask value A_3m' is uniformly distributed in $GF(2^n)$ because the matrix A_3 is invertible hence it induces a permutation. Thus, the proof of this lemma is completed by lemma 1. □

VI. Conclusion

This paper proposes an efficient masked implementation of the SEED algorithm using a new-style masked S-box. The proposed masked S-box can reduce a large number of operations of the masking addition as well as the RAM usage of the masked S-box. The results of the performance analysis indicate that the proposed masked implementation of SEED

reduces the RAM requirements from 512 bytes to 288 bytes. It also reduces the processing time by 38% compared with the masked SEED using the general masked S-box. The proposed method can be applicable to some block ciphers that have S-boxes and modular additions as nonlinear components, such as MARS, GOST, and BLOWFISH [17]-[19].

References

[1] P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," *CRYPTO*, Springer-Verlag, 1999, pp. 388-397.

[2] C. Herbst, E. Oswald, and S. Mangard, "An AES Smart Card Implementation Resistant to Power Analysis Attacks," *ACNS*, *LNCS*, Springer-Verlag, vol. 3989, 2006, pp. 239-252.

[3] E. Oswald and K. Schramm "An Efficient Masking Scheme for AES Software Implementations," *WISA*, *LNCS*, Springer-Verlag, vol. 3786, 2006, pp. 292-305.

[4] E. Trichina, D.S. Seta, and L. Germani, "Simplified Adaptive Multiplicative Masking for AES," *CHES*, *LNCS*, Springer-Verlag, vol. 2523, 2003, pp. 71-85.

[5] J. Blömer, J. Guajardo, and V. Krummel. "Provably Secure Masking of AES," *SAC*, *LNCS*, Springer-Verlag, vol. 3357, 2005, pp. 69-83.

[6] K. Schramm and C. Paar, "Higher Order Masking of the AES," *CT-RSA*, *LNCS*, Springer-Verlag, vol. 3860, 2006, pp. 208-225.

[7] M.L. Akkar and C. Giraud, "An Implementation of DES and AES, Secure against Some Attacks," *CHES*, *LNCS*, Springer-Verlag, vol. 2162, 2001, pp. 309-318.

[8] T. Messerges, "Securing the AES Finalists against Power Analysis Attacks," *FSE*, *LNCS*, Springer-Verlag, vol. 1978, 2001, pp. 293-301.

[9] Korea Internet & Security Agency, "Block Cipher Algorithm SEED." Available at: http://seed.kisa.or.kr/eng/about/about.jsp

[10] D. Kwon et al., "New Block Cipher: ARIA," *ICISC*, *LNCS*, Springer-Verlag, vol. 2971, 2004, pp. 432-445.

[11] J. Daemen and V. Rijmen, *The Design of Rijndael: AES - The Advanced Encryption Standard*, Springer-Verlag, 2002.

[12] L. Goublin, "A Sound Method for Switching between Boolean and Arithmetic Masking," *CHES*, *LNCS*, Springer-Verlag, vol. 2162, 2001, pp. 3-15.

[13] J.S. Coron and A. Tchulkine, "A New Algorithm for Switching from Arithmetic to Boolean Masking," *CHES*, *LNCS*, Springer-Verlag, vol. 2779, 2003, pp. 89-97.

[14] O. Neiße and J. Pulkus, "Switching Blindings with a View Towards IDEA," *CHES*, *LNCS*, Springer-Verlag, vol. 3156, 2004, pp. 125-133.

[15] H.S. Kim et al., "Efficient Masking Methods Appropriate for the Block Ciphers ARIA and AES," *ETRI J.*, vol. 32, no. 3, June 2010, pp. 370-379.

[16] Atmel Corporation. Datasheet: ATmega128(L). Available at: http://www.atmel.com/products/avr/

[17] E. Biham and V. Furman, "Impossible Differential on 8-Round MARS Core," NESSIE, NES/DOC/TEC/WP3/001/1, Sept. 11, 2000.

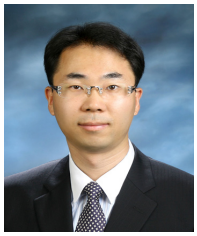
[18] GOST, Gosudarstvennyi Standard 28147-89, "Cryptographic Protection for Data Processing Systems," Government Committee of the USSR for Standards, 1989.

[19] B. Schneier, "Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish)," *FSE*, *LNCS*, Springer-Verlag, vol. 809, 1993, pp. 191-204.

HeeSeok Kim received his BS in mathematics from Yonsei University, Seoul, Rep. of Korea, in 2006 and the MS 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 and public-key and symmetric-key cryptosystems.

Young In Cho received the BS in mathematics from Hanyang University, Korea, in 2006 and the MS of engineering in information security from Korea University, Korea, in 2009. She is currently a PhD student with Graduate School of Information Management and Security, Korea University, and also a researcher of Center for Information Security Technologies (CIST). Her research interests include algorithms and architectures for computations in Galois fields and fast implementation of pairing cryptosystems.

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. He has been a senior researcher with ETRI, Daejeon, Rep. of Korea, since January 2002. 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 the editor of the ITU-T Rec. X.1171.



Dong-Guk Han received his BS in mathematics from Korea University in 1999, and his MS 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, Japan, from April 2004 to March 2005. He was a senior researcher in ETRI, Daejeon, Korea, from June 2006 to February 2009. He is currently working as an assistant professor with the Department of Mathematics of Kookmin University, Seoul, Rep. of Korea. He is a member of KIISC, IEEK, and IACR.

Seokhie Hong received his MA and PhD in mathematics from Korea University in 1997 and 2001, respectively. He had worked in SECURITY Technologies Inc. from 2000 to 2004. From 2004 to 2005, he worked as a post doctoral researcher in COSIC at K.U. 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.