A Fast and Provably Secure Higher-Order Masking of AES S-Box*

HeeSeok Kim, Seokhie Hong, and Jongin Lim

Center for Information Security Technologies, Korea University, Korea
{80khs,shhong,jilim}@korea.ac.kr
http://cist.korea.ac.kr

Abstract. This paper proposes an efficient and secure higher-order masking algorithm for AES S-box that consumes the most computation time of the higher-order masked AES. During the past few years, much of the research has focused on finding higher-order masking schemes for this AES S-box, but these are still slow for embedded processors use. Our proposed higher-order masking of AES S-box is constructed based on the inversion operation over the composite field. We replace the subfield operations over the composite field into the table lookup operation, but these precomputation tables do not require much ROM space because these are the operations over $GF(2^4)$. In the implementation results, we show that the higher-order masking scheme using our masked S-box is about 2.54 (second-order masking) and 3.03 (third-order masking) times faster than the fastest method among the existing higher-order masking schemes of AES.

Keywords: AES, side channel attack, higher-order masking, higher-order DPA, differential power analysis.

1 Introduction

Since Kocher introduced the concept of differential power analysis (DPA) [13], the security of block ciphers has received considerable attention, and it is now obvious that the unprotected implementations of block ciphers in embedded processors can be broken by DPA. 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) [3,5,9,11,15,16].

Recently, the important effort has been carried out to find a masking method that is secure against the higher-order DPA (HODPA) [14,22] as well as FODPA [22,17,18]. These masking schemes are called the higher-order masking schemes. Also, the higher-order masking scheme to counteract d-th order DPA [22] is

* "This research was supported by the MKE(The Ministry of Knowledge Economy), Korea, under the "itrc" support program supervised by the NIPA(National IT Industry Promotion Agency)" (NIPA-2011-C1090-1001-0004).

B. Preneel and T. Takagi (Eds.): CHES 2011, LNCS 6917, pp. 95–107, 2011.

called the d-th order masking scheme[1]. In this d-th order masking scheme, every intermediate value I of an original cipher is randomly split into (d+1)-tuple (I_0, I_1, ..., I_d). Here, for any intermediate value I, there exists a group operation $\perp$ such that $\perp(I_0, I_1, ..., I_d)$ equals I. This randomly split (d+1)-tuple can block that the combination of d or less elements in this tuple is dependent on the intermediate value I. Thus, security against d-th order DPA can be provided.

In the higher-order masking scheme of standard block cipher AES [1], the most important part is the S-box operation, which is the only non-linear operation of AES. Actually, most of the cost for higher-order masked AES is required by this non-linear part. Thus, to construct the higher-order masking scheme of AES in all previous works, the most important consideration has been to mask this S-box operation. To mask this operation, the initial works of [22] and [17] carry out table re-computation before all S-box operation. However, these methods require much computation time. M. Rivain and E. Prouff introduced a higher-order masked S-box operation based on the exponentiation operation to solve this problem [18]. This method can considerably reduce computation time compared with the existing methods. However, this scheme is still about 60 (second-order masking) and 130 (third-order masking) times slower than the straightforward AES.

In this paper, we propose a new higher-order masking of AES S-box based on the inversion operation over the composite field [20,21]. This method uses the precomputation tables for subfield operations such as the multiplication, square, and scalar multiplication over $GF(2^4)$. These tables can considerably reduce the time required. Also, because these tables are used for the operations over $GF(2^4)$, our method does not require much ROM space. The security of this new algorithm can be easily proved via the proofs in [18]. In the implementation results, our method is about 2.54 (second-order masking) and 3.03 (third-order masking) times faster than the method in [18]. Also, to use the higher-order masking scheme in embedded processors, we show the implementation results that apply the higher-order masking scheme to the first two and the last two rounds only, and the first-order masking to the other rounds and key-schedule. This is because HODPA generally attacks the first and last few rounds. Implementation results for this reduced masking are just 8.6 (second-order masking) and 13.8 (third-order masking) times slower than the straightforward AES. These numerical values mean that the reduced masking using our higher-order masked AES S-box can be sufficiently used in embedded processors.

The remainder of this paper is organized as follows. Section 2 describes the higher-order masking of AES and the inversion operation over the composite field. In Section 3, we introduce the new higher-order masking of AES S-box. Section 4 simply demonstrates the security of our method based on the proofs in [18] and Section 5 shows its efficiency. Finally, in Section 6, we offer the conclusion.

[1] Since the method of [22] has been known insecure for $d \geq 3$ [7], the method of [18] is currently the only higher-order masking scheme for $d \geq 3$.

2 Preliminaries

2.1 Advanced Encryption Standard (AES)

The Advanced Encryption Standard (AES), also known as Rijndael, is a block cipher adopted as an encryption standard by the US government [1]. This block cipher is composed of an SPN structure [4,6]. For an N-bit SPN-type block cipher of r rounds, each round consists of three layers. These layers are the key mixing layer, substitution layer, and linear transformation layer. In the key mixing layer, the round input is bitwise exclusive-ORed 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 a non-linear bijective mapping $\pi : \mathbb{F}_{2^n} \rightarrow \mathbb{F}_{2^n}$. In the case of AES, an S-box fulfills the role of this bijective mapping π. AES S-box is defined by a multiplicative inverse $b = a^{-1}$ in $GF(2^8)$ (except if $a = 0$ then $b = 0$) and an affine transformation as in the following equations:

$$\begin{array}{rcccc} S & : & GF(2^8) & \rightarrow & GF(2^8) \\ & & x & \rightarrow & Mx^{254} \oplus v \end{array}$$

where the value of x^{254} is regarded as a $GF(2)$-vector of dimension 8, M is an $8{\times}8$ $GF(2)$-matrix, and v is an $8{\times}1$ $GF(2)$-vector. The resulting value of the substitution layer becomes the N bit input $(i_0, i_1, , i_{N-1})$ 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 $O = LI$, where L is an $N{\times}N$ matrix and I is $(i_0, i_1, , i_{N-1})^T$. In AES, ShiftRows and MixColumns play this role. The linear transformation layer is omitted from the last round since it is easily shown that its inclusion adds no cryptographic strength.

2.2 Higher-Order Masking of AES

A d-th order masking method e' for an encryption algorithm $c \leftarrow e(m, k)$ is defined as follows, where m, k and c are plaintext, key and ciphertext, respectively [18]:

$$(c_0, c_1, ..., c_d) \leftarrow e'((m_0, m_1, ..., m_d), (k_0, k_1, ..., k_d))$$

In the equation above, d-th order masking method e' must satisfy the equations of $c = \perp_{i=0}^{d} c_i$, $m = \perp_{i=0}^{d} m_i$ and $k = \perp_{i=0}^{d} k_i$, where $\perp$ is the specific group operation. In this paper, we consider that this group operation is the exclusive-or (XOR, $\oplus$).

To guarantee security against d-th order DPA which exploits the leakages related to d intermediate values [22], the intermediate value I of the encryption algorithm e must also be replaced into $(I_0, I_1, ..., I_d)$. Here, the sum of d or less intermediate values $\bigoplus_{i \in S} I_i$ is independent of the sensitive data value (the intermediate value of the original cipher) I where S is a subset of $\{0, 1, ..., d\}$ and $1 \leq size(S) \leq d$.

M. Rivain and E. Prouff introduced the d-th order masking scheme for AES satisfying the conditions above. In this scheme, the higher-order masking method can be easily applied to every operation, except S-box, because these operations are linear operations. Namely, the linear operation $O \leftarrow L(I)$ can be replaced into $(O_0, O_1, ..., O_d) \leftarrow L'((I_0, I_1, ..., I_d))$ where $O_i = L(I_i)$, because the operation L' satisfies the equation of $\bigoplus_{i=0}^{d} O_i = L(\bigoplus_{i=0}^{d} I_i)$.

However, it is not easy to apply the higher-order masking to the S-box operation. Actually, the higher-order masking scheme spends most of the time for computing this non-linear operation. In the initial works on the higher-order masking method [22,17], it is general to carry out table re-computation before S-box operation, but this operation consumes a lot of time.

To reduce the time required, M. Rivain and E. Prouff introduced the d-th order secure **SecSbox** operation based on the exponentiation operation [18]. They found the following addition chain that can minimize not the number of squares but the number of multiplications because the square is a linear operation.

$$x \xrightarrow{S} x^2 \xrightarrow{M} x^3 \xrightarrow{2S} x^{12} \xrightarrow{M} x^{15} \xrightarrow{4S} x^{240} \xrightarrow{M} x^{252} \xrightarrow{M} x^{254}$$

In the chain above, S, M, $2S$ and $4S$ mean the square, multiplication, two squares and four squares, respectively. Here, the square, two squares and four squares are the linear operations. Thus, these operations are easily implemented using the look-up tables (LUTs). However, the other four multiplications must be carefully constructed because these are non-linear operations. They designed the d-th order secure multiplication algorithm **SecMult** using the Ishai-Sahai-Wagner (ISW) scheme [10]. This algorithm is described in Algorithm 1.

Algorithm 1. SecMult function [18]

Input: two $(d+1)$-tuples $(a_0, a_1, ..., a_d)$, $(b_0, b_1, ..., b_d)$ where $\bigoplus_{i=0}^{d} a_i = a$, $\bigoplus_{i=0}^{d} b_i = b$
Output: $(d+1)$-tuple $(c_0, c_1, ..., c_d)$ satisfying $\bigoplus_{i=0}^{d} c_i = ab$

1. For $i = 0$ to d do
   (a) For $j = i + 1$ to d do
      i. $r_{i,j} \leftarrow$ **rand**(8)
      ii. $r_{j,i} \leftarrow (r_{i,j} \oplus a_i b_j) \oplus a_j b_i$
2. For $i = 0$ to d do
   (a) $c_i \leftarrow a_i b_i$
   (b) For $j = 0$ to d, $j \neq i$ do, $c_i \leftarrow c_i \oplus r_{i,j}$

2.3 The Inversion Operation over a Composite Field

In order to reduce the cost of AES S-box, inversion methods over a composite field have been proposed [20,21]. These methods transform an element over $GF(2^8)$ into an element over the composite field having low inversion cost by the isomorphism function δ, 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 δ^{-1} into the element over $GF(2^8)$.

In [21], the composite field is built by repeating degree-2 extensions with the following irreducible polynomials:

$$\begin{aligned} GF(2^2) &: P_0(x) = x^2 + x + 1, \text{ where } P_0(\alpha) = 0, \\ GF((2^2)^2) &: P_1(x) = x^2 + x + \alpha, \text{ where } P_1(\beta) = 0, \\ GF(((2^2)^2)^2) &: P_2(x) = x^2 + x + \lambda, \text{ where } \lambda = (\alpha + 1)\beta,\ P_2(\gamma) = 0. \end{aligned}$$

Two isomorphism functions δ and δ^{-1} according to the above irreducible polynomials are as follows:

$$\begin{aligned} \delta &: GF(2^8) \rightarrow GF(((2^2)^2)^2) \\ \delta^{-1} &: GF(((2^2)^2)^2) \rightarrow GF(2^8) \end{aligned}$$

$$\delta : \begin{pmatrix} 1&1&0&0&0&0&1&0 \\ 0&1&0&0&1&0&1&0 \\ 0&1&1&1&1&0&0&1 \\ 0&1&1&0&0&0&1&1 \\ 0&1&1&1&0&1&0&1 \\ 0&0&1&1&0&1&0&1 \\ 0&1&1&1&1&0&1&1 \\ 0&0&0&0&0&1&0&1 \end{pmatrix} \qquad \delta^{-1} : \begin{pmatrix} 1&0&1&0&1&1&1&0 \\ 0&0&0&0&1&1&0&0 \\ 0&1&1&1&1&0&0&1 \\ 0&1&1&1&1&1&0&0 \\ 0&1&1&0&1&1&1&0 \\ 0&1&0&0&0&1&1&0 \\ 0&0&1&0&0&0&1&0 \\ 0&1&0&0&0&1&1&1 \end{pmatrix}$$

The inversion operation of an input value $A = a_h\gamma + a_l \in GF(((2^2)^2)^2)$, where a_h and a_l are elements in $GF((2^2)^2)$, is performed as follows in this composite field.

First, $A^{-1} \in GF(((2^2)^2)^2)$ is computed by $C^{-1}A^{16}$ ($C = A^{17} \in GF((2^2)^2)$). 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 $\lambda a_h^2 + (a_h + a_l)a_l$ due to $\gamma^2 + \gamma = \lambda$. After computing the inverse of A^{17} over $GF((2^2)^2)$, the computation of $A^{-1} = C^{-1}A^{16}$ can be completed. Figure 1 represents the AES S-box operation including the inversion operation over the composite field $GF(((2^2)^2)^2)$ where A_f is an affine transformation. For additional operations over both fields $GF((2^2)^2)$ and $GF(2^2)$ refer to [21].

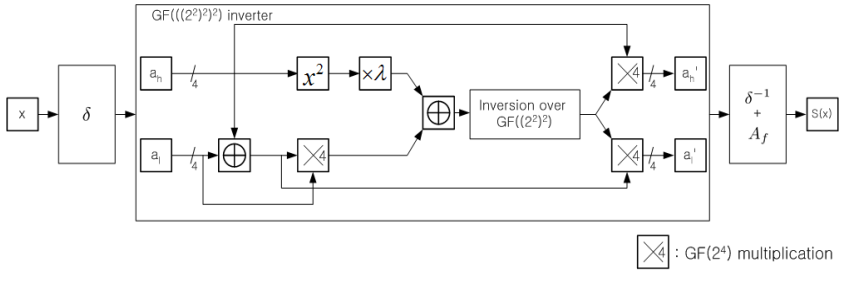

Fig. 1. S-box operation of AES

In Fig. 1, an equation for computing the output value $S(x)$ of x is performed as follows:

Step 1. $a_h\gamma + a_l = \delta(x)$ where a_h and a_l are elements in $GF((2^2)^2)$
Step 2. $d = \lambda a_h^2 + a_l(a_h + a_l) \in GF((2^2)^2)$
Step 3. $d' = d^{-1} \in GF((2^2)^2)$
Step 4. $a'_h = d'a_h \in GF((2^2)^2)$
Step 5. $a'_l = d'(a_h + a_l) \in GF((2^2)^2)$
Step 6. $S(x) = A_f(\delta^{-1}(a'_h\gamma + a'_l)) \in GF(((2^2)^2)^2)$

3 A Fast and Provably Secure Higher-Order Masking of AES S-Box

In this section, we propose a fast and provably secure higher-order masking of AES S-box. As aforementioned, the higher-order masking scheme of AES spends the most time computing the masked S-box operation. The main purpose of this paper is to reduce running time of the higher-order masking algorithm. Thus, our masked S-box uses the six precomputation tables. Most of the elements of these tables are the 4-bit data, but we allocate one byte for each element to simplify accessing the table elements. These precomputation tables are as follows with the notations defined in Section 2.3:

1. Squaring table $T1$ (The requirement for 16 bytes of ROM)
   - **Input :** $X \in GF(2^4)$
   - **Output :** $T1[X] = X^2 \in GF(2^4)$
2. Two squaring table $T2$ (The requirement for 16 bytes of ROM)
   - **Input :** $X \in GF(2^4)$
   - **Output :** $T2[X] = X^4 \in GF(2^4)$
3. Squaring-scalar multiplication table $T3$ (The requirement for 16 bytes of ROM)
   - **Input :** $X \in GF(2^4)$
   - **Output :** $T3[X] = \lambda X^2 \in GF(2^4)$
4. Multiplication table $T4$ (The requirement for 256 bytes of ROM)
   - **Input :** $X, Y \in GF(2^4)$
   - **Output :** $T4[X][Y] = XY \in GF(2^4)$
5. Isomorphism table $T5$ (The requirement for 256 bytes of ROM)
   - **Input :** $X \in GF(2^8)$
   - **Output :** $T5[X] = \delta(X) \in GF(((2^2)^2)^2)$
6. Inverse isomorphism-Affine transformation table $T6$ (The requirement for 256 bytes of ROM)
   - **Input :** $X \in GF(((2^2)^2)^2))$
   - **Output :** $T6[X] = A_f(\delta^{-1}(X)) \in GF(2^8)$

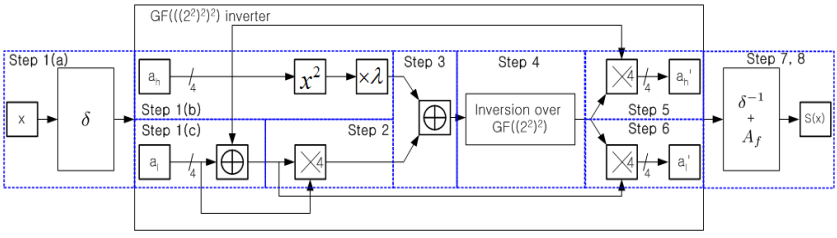

Fig. 2. The eight steps of the proposed masked S-box

As aforementioned, the d-th order masking scheme must compute every intermediate value I of the original encryption algorithm e with the form of $(I_0, I_1, ..., I_d)$ where $\bigoplus_{i=0}^{d} I_i = I$. We design a new higher-order masked S-box satisfying this condition in each step over the composite field. We first classify the S-box operation of Fig. 1 into the nine steps of Fig. 2.

In Fig. 2, the operations of Step 1(a), 1(b), 7 and 8 are the linear operations or affine transformation. Here, the linear operation $O \leftarrow L(I)$ can be easily replaced into $(O_0, O_1, ..., O_d) \leftarrow L'((I_0, I_1, ..., I_d))$, where $O_i = L(I_i)$, because the operation L' satisfies the equation of $\bigoplus_{i=0}^{d} O_i = L(\bigoplus_{i=0}^{d} I_i)$. Affine transformation $O \leftarrow A_f(I)$ can also be replaced into $(O_0, O_1, ..., O_d) \leftarrow A'_f((I_0, I_1, ..., I_d))$, where $O_0 = A_f(I_0) \oplus (0x63 \times (d \ mod \ 2))$ and $O_i = A_f(I_i)(i \neq 0)$, because the operation A'_f satisfies the equation of $\bigoplus_{i=0}^{d} O_i = A_f(\bigoplus_{i=0}^{d} I_i)$. We show hereafter some methods to apply the d-th order masking to the remaining non-linear operations.

- **Masking XOR operation:** The masked XOR operation can be easily constructed. This operation outputs $(x_0 \oplus y_0, x_1 \oplus y_1, ..., x_d \oplus y_d)$ from two input $(d+1)$-tuples $(x_0, x_1, ..., x_d)$, $(y_0, y_1, ..., y_d)$. Here, two input $(d+1)$-tuples must be independent of each other as mentioned in [18].

- **Masking $GF(2^4)$ multiplication:** We construct the masked $GF(2^4)$ multiplication using Algorithm 1 of [18]. This algorithm is described in Algorithm 2. The only difference between two algorithms is whether or not the multiplication table is used. Our masked S-box computes the inversion by using the subfield operation over $GF(2^4)$. Thus, most operations, including $GF(2^4)$ multiplication, can be computed by using the precomputation tables. These require ROM space, but the size required is very small.

- **Masking $GF(2^4)$ inversion:** The masking method for $GF(2^4)$ inversion can be designed variously. One way is to use the composite field operation over $GF((2^2)^2)$ similarly to the masked operation over $GF(((2^2)^2)^2)$. However, this method requires as many table lookup operations as that over $GF(((2^2)^2)^2)$. Therefore, we use the operation of x^{14} over $GF(2^4)$. The addition chain of x^{14} to minimize the number of multiplications can be constructed as follows:

$$x \xrightarrow{S} x^2 \xrightarrow{M} x^3 \xrightarrow{2S} x^{12} \xrightarrow{M} x^{14}$$

The masked inversion algorithm over $GF(2^4)$ using this addition chain is shown in Algorithm 3. In Algorithm 3, for **RefreshMasks** algorithm to eliminate the dependence between the two input tuples of **SecMult4** function refer to [18].

Algorithm 2. $GF(2^4)$ **SecMult4** function using the multiplication table $T4$

Input: two $(d+1)$-tuples $(a_0, a_1, ..., a_d)$, $(b_0, b_1, ..., b_d)$ where $\bigoplus_{i=0}^{d} a_i = a$, $\bigoplus_{i=0}^{d} b_i = b$
Output: $(d+1)$-tuple $(c_0, c_1, ..., c_d)$ satisfying $\bigoplus_{i=0}^{d} c_i = ab \in GF(2^4)$

1. For $i = 0$ to d do
   (a) For $j = i + 1$ to d do
      i. $r_{i,j} \leftarrow$ **rand**(4)
      ii. $r_{j,i} \leftarrow (r_{i,j} \oplus T4[a_i][b_j]) \oplus T4[a_j][b_i]$
2. For $i = 0$ to d do
   (a) $c_i \leftarrow T4[a_i][b_i]$
   (b) For $j = 0$ to d, $j \neq i$ do, $c_i \leftarrow c_i \oplus r_{i,j}$

Algorithm 3. $GF(2^4)$ **SecInv** function

Input: $(d+1)$-tuple $(x_0, x_1, ..., x_d)$ satisfying $\bigoplus_{i=0}^{d} x_i = x \in GF(2^4)$
Output: $(d+1)$-tuple $(y_0, y_1, ..., y_d)$ satisfying $\bigoplus_{i=0}^{d} y_i = x^{-1} = x^{14} \in GF(2^4)$

1. For $i = 0$ to d do
   (a) $w_i = T1[x_i]$
2. **RefreshMasks**$((w_0, w_1, ..., w_d))$
3. $(z_0, z_1, ..., z_d)$=**SecMult4**$((w_0, w_1, ..., w_d), (x_0, x_1, ..., x_d))$
4. For $i = 0$ to d do
   (a) $z_i = T2[z_i]$
5. $(y_0, y_1, ..., y_d)$=**SecMult4**$((z_0, z_1, ..., z_d), (w_0, w_1, ..., w_d))$

Algorithm 4 presents the entire operation of the proposed d-th order masking for AES S-box. The meaning of the operations carried out in each step is described in Fig. 2.

4 Security Analysis

The security of the proposed algorithm can be easily proved by the proofs in [18]. It is straightforward to prove the security for all operations, except the **SecMult4** algorithm because each element of the input tuple is independently operated in these operations. Also, the security of **SecMult4** algorithm can be

proved by Theorem 1 of [18]. The remaining consideration is the independence between two input $(d+1)$-tuples of the **SecMult4** algorithm. In Algorithm 4, two input tuples of **SecMult4** algorithm are independent of each other. However, in Step 3 of Algorithm 3, the input tuples of **SecMult4** function (x, x^2) have the dependency. However, **RefreshMasks** in Step 2 can eliminate this dependency as mentioned in Section 3. Thus, the proposed algorithm provides security against the d-th order DPA, that is, it can guarantee that every combination of d or less intermediate values is independent of any sensitive data value.

Algorithm 4. d-th order masking of AES S-box

Input: $(d+1)$-tuple $(x_0, x_1, ..., x_d)$ satisfying $\bigoplus_{i=0}^{d} x_i = x \in GF(2^8)$
Output: $(d+1)$-tuple $(y_0, y_1, ..., y_d)$ satisfying $\bigoplus_{i=0}^{d} y_i = Sbox(x) \in GF(2^8)$

1. For $i = 0$ to d do
   (a) $(H_i||L_i) = T5[x_i]$ /*H_i, $L_i \in GF(2^4)$*/
   (b) $w_i = T3[H_i]$
   (c) $t_i = H_i \oplus L_i$
2. $(L_0, L_1, ..., L_d)$=**SecMult4**$((t_0, t_1, ..., t_d), (L_0, L_1, ..., L_d))$
3. For $i = 0$ to d do
   (a) $w_i = w_i \oplus L_i$
4. $(w_0, w_1, ..., w_d)$=**SecInv**$((w_0, w_1, ..., w_d))$
5. $(H_0, H_1, ..., H_d)$=**SecMult4**$((w_0, w_1, ..., w_d), (H_0, H_1, ..., H_d))$
6. $(L_0, L_1, ..., L_d)$=**SecMult4**$((w_0, w_1, ..., w_d), (t_0, t_1, ..., t_d))$
7. For $i = 0$ to d do
   (a) $y_i = T6[H_i||L_i]$
8. If d is odd, $y_0 = y_0 \oplus 0x63$
9. Return $(y_0, y_1, ..., y_d)$.

5 Performance Analysis and Implementation Results

In our d-th order masked S-box, **SecMult4** function requires $(d+1)^2$ table lookup operations, $2d(d+1)$ XOR operations, and the generation of $2d(d+1)$ random bits. Considering 5 **SecMult4** function calls and other minor operations (**RefreshMasks**, table lookup operations, and 4-bit shift operations[2]), our masked S-box requires totally $(5d^2 + 13d + 8)$ table lookup operations, $(10d^2 + 16d + 5)$ XOR operations, $(\frac{10d^2+14d}{8} + 2(d+1))$ 4-bit shift operations, $(\frac{10d^2+14d}{8} + 2(d+1))$ bitwise AND operations and the generation of $(10d^2 + 14d)$ random bits[3].

[2] 4-bit shift operation may require 4 instruction calls unless the single instruction carrying out 4-bit shift is supported. However, some microcontrollers like 8051 and AVR family support a single SWAP operation, which swaps high and low nibbles in a register.

[3] To get the random nibbles from $(10d^2 + 14d)$ random bits, we split 1 random byte into two nibbles. This method requires one 4-bit shift operation and one bitwise AND operation. Therefore, the generation of $(10d^2 + 14d)$ random bits involves $(\frac{10d^2+14d}{8})$ 4-bit shift and bitwise AND operations.

Table 1. Comparison of two d-th order masked S-box schemes in terms of the total number of operations

	Ours	[18]
Table Lookup	$5d^2 + 13d + 8$	$12d^2 + 31d + 19$
XOR	$10d^2 + 16d + 5$	$8d^2 + 12d$
Random Bits	$10d^2 + 14d$	$16d^2 + 32d$
etc	4-bit logical shift : $\frac{5}{4}d^2 + \frac{15}{4}d + 2$, 8-bit bitwise AND : $\frac{5}{4}d^2 + \frac{15}{4}d + 2$	8-bit Addition : $8(d+1)^2$, 8-bit logical AND : $4(d+1)^2$

On the other hand, the d-th order masked S-box in [18] involves 4 **SecMult** function calls, i.e., $4(d+1)^2$ multiplications over $GF(2^8)$. The multiplications over $GF(2^8)$ can be efficiently implemented with $log/alog$ tables (see Appendix A). Here, we remove the conditional branching operation because this operation leaks some information that can be exploited by simple power analysis (SPA) [13]. Also, we remove the reduction operation modulo 255 to improve the computation speed. 4 **SecMult** function calls require $12(d+1)^2$ table lookup operations because one multiplication over $GF(2^8)$ involves 3 table lookup operations. Table 1 compares two d-th order masked S-box schemes in terms of the total number of operations.

To compare the performance of our masked S-box with the existing countermeasures in embedded processors, we use Algorithms 6 and 7 in [19]. Here, we replace the masked S-box operation of these algorithms into our algorithm.

We implement AES-128 in C-language for ATmega128 8-bit architecture [2]. First, the straightforward AES requires 11,170 clock cycles. We implement the first-order masking of AES using the method in [9], but we do not apply the dummy operation and the shuffling method, which provide partial security against the second-order DPA. This requires 19,525 clock cycles.

In the implementation of the second-order masking methods, we compare our method with the methods in [17] and [18]; [18] is the most recent work on the higher-order masking of AES. Our method requires slightly more ROM size than the method in [18], but is 2.54 times faster and also 4.51 times faster than the method in [17].

However, our method is still 23 times slower than the straightforward AES. Thus, we consider the additional case for applying the second-order masking to only the first two and the last two rounds. This is because the second-order DPA generally attacks the first and last few rounds. Also, we apply the first-order masking to the key-schedule and the rest of the rounds (3∼8 rounds). After finishing the key-schedule operation, we change the first two and the last two round keys into the form of $(d+1)$-tuple by using d random numbers. To provide security against the analysis such as [8] and [12], we apply the first-order masking to 3∼8 rounds. The implementation result is just 8.6 times slower than the straightforward AES. These numerical values mean that this algorithm can be used practically in the embedded processors.

Method	Cycles (x 10^3 cc)			Table Size (Bytes)	
	KeyExpand	Encryption	Total	RAM	ROM
Original AES					
No Masking (Straightforward AES)	2.2	9.0	11.2	0	256
First Order Masking					
ACNS'06 [9] (No dummy, No Shuffling)	4.6	14.9	19.5	256	256
Second Order Masking					
FSE'08 [17] (Complete second-order masking)	247.4	950.0	1197.4	256	256
CHES'10 [18] (Complete second-order masking)	144.1	531.2	675.4	0	768
Ours (Complete second-order masking)	66.2	199.3	265.5	0	816
Ours (KeyExpand (first) Enc. (1,2,9,10:second, 3~8:first))	5.2	90.6	95.8	256	1062
Third Order Masking					
CHES'10 [18] (Complete third-order masking)	293.4	1102.9	1396.3	0	768
Ours (Complete third-order masking)	114.6	346.8	461.3	0	816
Ours (KeyExpand (first) Enc. (1,2,9,10:third, 3~8:first))	5.5	149.6	155.1	256	1062

We also implement the third-order masking, and compare it with the method of [18]. Here, our method is 3.03 times faster than the existing countermeasure and 41.03 times slower than the straightforward AES. Also, the reduced masking is just 13.8 times slower than the straightforward AES.

6 Conclusion

In this paper, we proposed a new higher-order masking method for AES S-box. Our method could considerably reduce the computation time of the higher-order masked AES. Our method was 2.54 times faster than the most recent method of the second-order masking, and it was 3.03 times faster than that of the third-order masking. Also, in order to use the second-order masking algorithm in embedded processors, we only applied the second (third) order masking to the first two and the last two rounds in the encryption of AES. The results for these implementations were just 8.6 (second) and 13.8 (third) times slower than the straightforward AES. These numerical values mean that our higher-order masked S-box can achieve practical use of the higher-order masked AES in embedded processors.

References

1. NIST, FIPS 197: Advanced Encryption Standard (2001)
2. Atmel Corporation: Datasheet: ATmega128(L), http://www.atmel.com/products/avr/
3. Akkar, M.-L., Giraud, C.: An Implementation of DES and AES, Secure against Some Attacks. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 309–318. Springer, Heidelberg (2001)
4. Adams, C., Tavares, S.: The Structured Design of Cryptographically Good SBoxes. Journal of Cryptology 3(1), 27–42 (1990)
5. Blömer, J., Guajardo, J., Krummel, V.: Provably Secure Masking of AES. In: Handschuh, H., Hasan, M.A. (eds.) SAC 2004. LNCS, vol. 3357, pp. 69–83. Springer, Heidelberg (2004)
6. O'Connor, L.: On the Distribution of Characteristics in Bijective Mappings. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 360–370. Springer, Heidelberg (1994)
7. Coron, J.-S., Prouff, E., Rivain, M.: Side Channel Cryptanalysis of a Higher Order Masking Scheme. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 28–44. Springer, Heidelberg (2007)
8. Handschuh, H., Preneel, B.: Blind Differential Cryptanalysis for Enhanced Power Attacks. In: Biham, E., Youssef, A.M. (eds.) SAC 2006. LNCS, vol. 4356, pp. 163–173. Springer, Heidelberg (2007)
9. Herbst, C., Oswald, E., Mangard, S.: An AES Smart Card Implementation Resistant to Power Analysis Attacks. In: Zhou, J., Yung, M., Bao, F. (eds.) ACNS 2006. LNCS, vol. 3989, pp. 239–252. Springer, Heidelberg (2006)
10. Ishai, Y., Sahai, A., Wagner, D.: Private Circuits: Securing Hardware against Probing Attacks. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 463–481. Springer, Heidelberg (2003)
11. Kim, H., Kim, T., Han, D., Hong, S.: Efficient Masking Methods Appropriate for the Block Ciphers ARIA and AES. ETRI Journal 32(3), 370–379 (2010)
12. Kim, J., Hong, S., Han, D., Lee, S.: Improved Side-Channel Attack on DES with the First Four Rounds Masked. ETRI Journal 31(5), 625–627 (2009)
13. Kocher, P.C., Jaffe, J., Jun, B.: Differential Power Analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)
14. Messerges, T.S.: Using Second-Order Power Analysis to Attack DPA Resistant Software. In: Paar, C., Koç, Ç.K. (eds.) CHES 2000. LNCS, vol. 1965, pp. 238–251. Springer, Heidelberg (2000)
15. Oswald, E., Mangard, S., Pramstaller, N., Rijmen, V.: A Side-Channel Analysis Resistant Description of the AES S-Box. In: Gilbert, H., Handschuh, H. (eds.) FSE 2005. LNCS, vol. 3557, pp. 413–423. Springer, Heidelberg (2005)
16. Oswald, E., Schramm, K.: An Efficient Masking Scheme for AES Software Implementations. In: Song, J.-S., Kwon, T., Yung, M. (eds.) WISA 2005. LNCS, vol. 3786, pp. 292–305. Springer, Heidelberg (2006)
17. Rivain, M., Dottax, E., Prouff, E.: Block Ciphers Implementations Provably Secure Against Second Order Side Channel Analysis. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 127–143. Springer, Heidelberg (2008)
18. Rivain, M., Prouff, E.: Provably Secure Higher-Order Masking of AES. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 413–427. Springer, Heidelberg (2010)

19. Rivain, M., Prouff, E.: Provably Secure Higher-Order Masking of AES, Cryptology ePrint Archive (2010), http://eprint.iacr.org/
20. Rudra, A., Dubey, P.K., Jutla, C.S., Kumar, V., Rao, J.R., Rohatgi, P.: Efficient Rijndael Encryption Implementation with Composite Field Arithmetic. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 171–188. Springer, Heidelberg (2001)
21. Satoh, A., Morioka, S., Takano, K., Munetoh, S.: A Compact Rijndael Hardware Architecture with S-Box Optimization. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 239–254. Springer, Heidelberg (2001)
22. Schramm, K., Paar, C.: Higher Order Masking of the AES. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 208–225. Springer, Heidelberg (2006)

A Multiplication in GF(2^8) without SPA Leakage

In [18], the higher-order masked S-box computes AES multiplications over $GF(2^8)$. The most efficient way for this operation is to use log and $alog$ tables where $log[\alpha^i] = i$ and $alog[i] = \alpha^i$ for a generator α of $GF(256)^*$ and $0 \leqslant i < 255$. Multiplication using these two tables is computed according to the following equation:

$$ab = \begin{cases} alog[(log[a] + log[b])\ mod\ 255] & \text{if both } a \text{ and } b \text{ are not zero} \\ 0 & \text{otherwise} \end{cases}$$

However, the reduction modulo 255 requires heavy computational cost and the conditional branching operation has to be removed to eliminate the possibility of SPA. We compute the reduction modulo 255 by using the reduction modulo 256 and remove the conditional branching operation. This algorithm is described in Algorithm 5. To reduce the number of operations, we also use $\overline{alog}$ table as in the following equation, which requires 1 byte more ROM size than $alog$ table.

$$\overline{alog}[x] = \begin{cases} alog[x] & \text{if } 0 \leqslant x < 255 \\ alog[0] & \text{if } x = 255 \end{cases}$$

Algorithm 5. Multiplication in $GF(2^8)$ without SPA Leakage

Input: a, $b \in GF(2^8)$, f is an irreducible polynomial over $GF(2^8)$
Output: $ab\ mod\ f$

1. $t = log[a]$
2. $s = (t + log[b])\ mod\ 2^8$
3. $r = \overline{alog}[(s < t) + s]$ /* $s < t$: the carry associated with Step 2 */
4. Return $(a\&\&b) * r$ /* &&: the logical AND operation */