International Journal of Computer Sciences and Engineering Systems
Vol. 12 No.1 (June, 2018)

A Secure against High Order DPA Attacks AES Coprocessor Based on Masking Scheme

Yuan-Man TONG, Zhi-Ying WANG, Kui DAI & Hong-Yi LU

School of Computer Science, National University of Defense TechnologyChangsha, Hunan 410073, China
E-mail: yuanmantong@yahoo.com.cn

Manuscript received December 5, 2017, Manuscript revised January 12, 2018

Abstract: *An AES implementation based on masking scheme is presented. Five masked primitive operations and their hardware circuits are defined. Then the masked data-path of all the transformations in AES is built on these primitive operations. All the input signals of the masked primitive operations are synchronized by the generated enable signals. And a simple method is proposed to generate random values to mask all the intermediate variables. Our scheme is probably secure against first order DPA (differential power analysis) and high order DPA (HODPA) attacks.*

Keywords: *side-channel attack, power analysis attack, AES, DPA, HODPA, masking scheme, primitive operation, probably security*

1. INTRODUCTION

Side-channel attacks, especially the power analysis attacks, on software or hardware implementations of various cryptosystems aim at recovering the secret key information from power consumption performed on the electronic device such as smartcard [1-2]. Power analysis attacks are very powerful as they do not require expensive resources. Almost all cryptographic algorithms including AES are susceptible to power analysis attacks while there are no appropriate countermeasures.

Power analysis attack is classified to SPA (simple power analysis), DPA (differential power analysis) and HODPA (high order DPA) [1-3]. SPA just exploits the relationship between the operations that are executed and the power leakage. DPA attacks use statistical analysis and error correction techniques to extract information correlated to secret keys. In fact, the DPA attack can be considered as a hypothesis testing including mean test and correlation analysis, the latter is also called CPA (Correlation Power Analysis) [4]. A HODPA attack is a generalization of the (first-order) DPA attack in which the power consumption curves are analyzed by using a joint statistic applied to collections of points in time.

The AES is the de-facto standard for symmetric encryption. And it is an attractive algorithm for many security relevant applications. Efficient implementation of AES on smartcards that is resistant to power analysis attacks is now a primary interest of the industry. Countermeasures for AES implementation in smartcards include masking scheme [5-17], random execution [10], and novel logic style [18-20] etc. The masking scheme is the widely used method to protect AES, and it can be implemented in software and in hardware.

This paper presents an AES implementation based on the masked primitive operations. Five masked primitive operations and their hardware circuits are defined. According to these masked primitive operations, the masked data-path of all the transformations in AES is established. To avoid the side-channel leakage pointed out in [21] caused by different arrival time, all input signals of the masked primitive operations are synchronized by generated enable signals. And a simple method is proposed to generate the random values to mask all the intermediate variables. The proposed AES implementation is provably secure against first order and high order DPA attacks.

2. RELATED WORKS

Masking scheme is a kind of frequently used algorithmic countermeasure against power analysis attacks. In a masked implementation, each intermediate variable is masked by a uniformly distributed random value. For instance, in Boolean masking, $X' = X \oplus R$, where X is masked by R and X' is the masked value. So the masked value never depends on the secret key and power analysis attack is prevented. However, some masked implementations do not really prevent power analysis attacks, i.e. they are vulnerable to DPA or HODPA. The following definition shows when an adversary can perform an n-th ($n \geq 1$) order DPA attacks [7, 22-23]. And we will use this definition to validate the resistance to power analysis of our method and other proposed masking schemes.

Definition GDPA (Generic DPA): Let $z_1, z_2, \ldots, z_n$ be n ($n \geq 1$) intermediate variables, and the joint distribution of $(z_1, z_2, \ldots, z_n)$ is denoted as $D(m, k)$ for a pair of (m, k) of plaintext and key, where $D(m, k)$ is discretely distributed.

If there exist at least two different assignments of (m, k) such that $D(m_1, k_1)$ is not identical with $D(m_2, k_2)$, then an attacker may be able to perform a n-th order DPA attack on ($z_1, z_2, \ldots, z_n$).

In the past, many different masking schemes have been presented to protect AES implementation.

(1) Multiplicative Masking Scheme

Akkar proposed the transformed multiplicative masking scheme for AES [5]. However, it has the so-called zero problem, i.e. zero is not masked at all. So the distributions of some intermediate variables are correlated to the key and satisfy the precondition of GDPA so that this scheme is vulnerable to DPA. Trichina and Goliæ then tried to enhance this scheme [9, 17]. But they did not solve the zero-problem perfectly.

(2) log/alog Tables based Masking

Trichina and Korkishko proposed a software-oriented masking method based on log/alog tables [16]. However, this scheme still has the zero-problem.

(3) Random Isomorphism based Masking

Rostovtsev and Shemyakina proposed to use isomorphism of the underlying finite field [14]. The first isomorphism in the paper has zero-problem too. For the second one, it must be carefully implemented or it may be subject to DPA attacks.

(4) Tower Fields Methods

Oswald proposed a masking scheme based on the arithmetic over GF(2-^{4}) [11-12]. In this scheme, the masks of the input and output of S-box are the same, so it is vulnerable to 2nd DPA. Blömer proposed the scheme based on the arithmetic over the composite field GF(((2^2)^2)^2) [7]. It has been proven to be true that this scheme is secure against DPA. However, it is pointed out that the glitches in masked multipliers lead to side-channel leakage [21].

(5) Fourier Transformation based Masking

Prouff proposed a Fourier transformation based masking scheme for AES S-box [13]. This scheme is proven to be secure against first order DPA, but it may be subject to 2nd DPA caused by the unsecure switching between arithmetic masking and Boolean masking.

(6) Algebraic Masking

Courtois proposed a general high-level algebraic method to protect AES against power attacks of any given order [8]. Using the group of homographic transformations over the projective space, a secure computation of a whole masked inverse is achieved. The high order DPA attacks resistant implementation seems to require large amount of computation.

(7) Masking and Randomization

Herbst proposed an efficient AES software implementation that is well suited for 8-bit smart cards and resistant against power analysis attacks [10]. This implementation masks the intermediate results and randomizes the sequence of operations at the beginning and the end of the AES execution. Resistant to high order DPA attacks mean that a large number of measurements are required for a successful attack. And the expected number of measurements is tunable.

(8) High order Masking

Schramm proposed a masking scheme which protects an AES implementation against n-th order DPA for any arbitrary chosen order n [15]. However, Coron pointed out that this scheme is subject to 3rd order DPA attacks [24].

So we can see that it is still an open problem to protect AES against high order DPA attacks.

3. MASKED PRIMITIVE OPERATIONS

The AES algorithm is a symmetric block cipher that operates on 128-bit data blocks. AES uses a cipher key to encrypt a 128-bit data block. The input, output and intermediate cipher result called State are represented as 4×4 arrays of bytes. As most symmetric ciphers, AES encrypts an input data-block by applying the same round function iteratively. In one round, the input state is mapped to the output state by performing the following four different transformations one after another.

(1) The **SubBytes** transformation is a non-linear byte substitution (which is called S-box) that operates independently on each byte of the State. For a byte x, the S-box on x, $S(x)$, is defined as follows

$$S(x) = L \cdot Inv(x) \oplus c, x \in \mathrm{GF}(2^8) \quad (1)$$

$$Inv(x) = \begin{cases} 0, x = 0 \\ x^{-1}, \text{otherwise} \end{cases} \quad (2)$$

where L and c are the parameters of the affine transformation in AES. $Inv(x)$ computes the inversion of x when x is not 0.

(2) In the **ShiftRows** transformation, the bytes in the last three rows of the State are cyclically shifted over different numbers of bytes (offsets). The first row is not shifted. In this paper, ShiftRows is not implemented directly, but integrated into MixColumns.

(3) **MixColumns** transforms each column of the State. Each byte in a column is interpreted as the coefficients of a polynomial in an extension field over GF(2^8). This polynomial is multiplied by the constant polynomial $c(x)$ = {03}x^3 + {01}x^2 + {01}x + {02}, where the coefficients are elements of GF(2^8) in hexadecimal notation. The modular remainder of the resulting product modulo $x^4 + 1$ is the output of **MixColumns**. In the last round, the MixColumns is not needed.

(4) In the **AddRoundKey** transformation, a round key is added to the State by a simple bitwise XOR operation (⊕). Each round key is derived from the key schedule and the size of a round key is equal to the State's. Before

the first round, the AddRoundKey is performed on the initial key and input block.

In this paper, $Inv(x)$ is implemented in the composite field $GF(((2^2)^2)^2)$ [7]. The following field structures are used:

$GF(2^2) \cong GF(2)[x]/P(x)$, $P(x) = (x^2 + x + 1)$, $P(\varphi) = 0$

$GF((2^2)^2) \cong GF(2^2)[y]/(y^2 + y + \varphi)$, $\varphi = (10)_2$,

$GF(((2^2)^2)^2) \cong GF((2^2)^2)[z]/(z^2 + z + \lambda)$, $\lambda = (1100)_2$

For $x \in GF(2^8)$, it is mapped into $GF(((2^2)^2)^2)$ by the isomorphism σ, i.e. $\sigma x \in GF(((2^2)^2)^2)$, where σ is a 8×8 binary matrix. After the $Inv(\sigma x)$ has been computed, it is then mapped to $GF(2^8)$ by the inverse isomorphism σ^{-1}, i.e. $Inv(x) = \sigma^{-1}(Inv(\sigma x))$. The isomorphism mapping σ is similar to the affine transformation. And σ can be combined with affine transformation, i.e. the new matrix $\sigma \times L$ is pre-computed and used.

Let $A \in GF(((2^2)^2)^2)$, then $Inv(A) = Inv(A \cdot A^{16}) \cdot A^{16}$ and $B = A^{17} \in GF((2^2)^2)$. $Inv(B) = Inv(B^5)B^4$ and $C = B^5 \in GF(2^2)$. And $Inv(C) = C^2$. The square and multiplication over the composite fields are computed as follows . Since A, B and C can be denoted by $A = az + b$, $a, b \in GF((2^2)^2)$, $B = cy + d$, $c, d \in GF(2^2)$, $C = ex + f$, $e, f \in GF(2)$, we get the following equations

$(az + b)^2 = a^2z + a^2\lambda + b^2$,

$(az + b)(a'z + b') = ((a + a')(b + b') + bb')z + aa'\lambda + bb'$,

$(cy + d)^2 = c^2y + c^2\varphi + d^2$,

$(cy + d)(c'y + d') = ((c + c')(d + d') + dd')y + cc'\varphi + dd'$,

$(ex + f)^2 = ex + (e + f)$,

$(ex + f)(e'x + f') = ((e + e')(f + f') + ff')x + ee' + ff'$.

The required computation can be further reduced using the following tricks.

- Since $B = A \cdot A^{16} \in GF((2^2)^2)$, the coefficient of z is not computed in the multiplication.

- B^4 can be computed as $cy + (c \oplus d)$, i.e., only one addition over $GF(2^2)$.

- Since $B^5 \in GF(2^2)$, it can be computed as $c^2\varphi \oplus cd \oplus d^2$.

All the transformations in AES algorithm can be decomposed to the following primitive operations. And these primitive operations are masked in this paper. Let $x_1 = u_1 \oplus r_1$, $x_2 = u_2 \oplus r_2$, $u_1, r_1, u_2, r_2 \in GF(2^8)$, where r_1 and r_2 are random values, and x_1 and x_2 are the masked variables.

(1) Addition (add)

The addition in AES is simple bit-wise XOR operation. The masked addition is defined as

$t_1 = (r_1 \oplus r_2) \oplus r'$

$t_2 = (x_1 \oplus x_2) \oplus t_1 = (u_1 \oplus u_2) \oplus r'$

In the masked addition, r' is a newly generated random value so that the desired result $(u_1 \oplus u_2)$ is masked by r'. The hardware circuit of masked addition is shown in Fig. 1.

(2) Affine Transformation (aff)

The affine transformation is the linear part of S-box. And the masked affine transformation is defined as

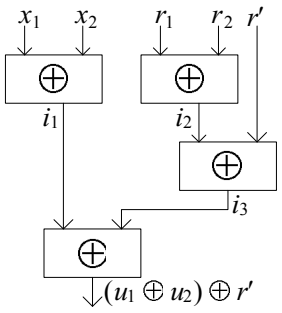

Figure 1: The Masked Addition

$$t_1 = L \cdot r_1 \oplus r'$$

$$t_2 = L \cdot x_1 \oplus c$$

$$t_3 = t_1 \oplus t_2$$

Similarly, the desired result $(L \cdot u_1 \oplus c)$ is also masked by r'. And the hardware circuit of masked affine transformation is shown in Fig. 2.

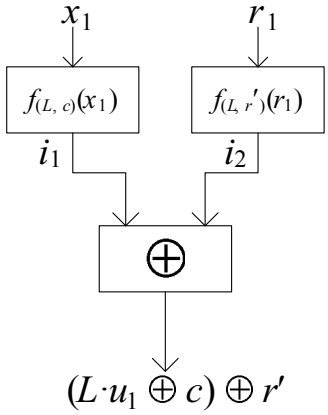

Figure 2: The Masked Affine Transformation

(3) Square (sqr)

While calculating $Inv(x)$ using the composite field arithmetic, square is the necessary operation. The masked square is defined as

$$t_1 = r_1^2 \oplus r'$$

$$t_2 = x_1^2 \oplus t_1 = u_1^2 \oplus r'$$

In the masked square, the desired result (u_1^2) is also masked by r'. And the hardware circuit is shown in Fig. 3.

(4) Multiplication (mul)

The masked multiplication is defined as

$t_1 = r_1r_2 \oplus r_3$

$t_2 = x_1r_2 \oplus t_1 \oplus (r_3 \oplus r_4)$

$t_3 = x_2r_1 \oplus t_2 \oplus (r_4 \oplus r_5)$

$t_4 = x_1x_2 \oplus t_3 \oplus (r_5 \oplus r2) = u_1u_2 + r'$

In the above computation, r_3, r_4, r_5, and r' are newly generated random values. And the desired result (u_1u_2) is masked by r'. And the hardware circuit is shown in Fig. 4.

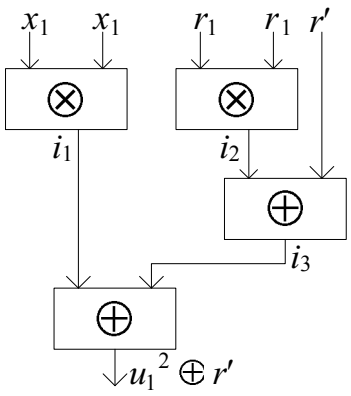

Figure 3: The Masked Square

(5) Multiplication with Constant Value (mulc)

When a variable x_1 is multiplied by a constant value, a, the masked multiplication is defined as

$$t_1 = ar_1 \oplus r'$$

$$t_2 = ax_1 \oplus t_1 = au_1 \oplus r'$$

Here the desired result (au_1) is also masked by r'. And the hardware circuit is shown in Fig. 5.

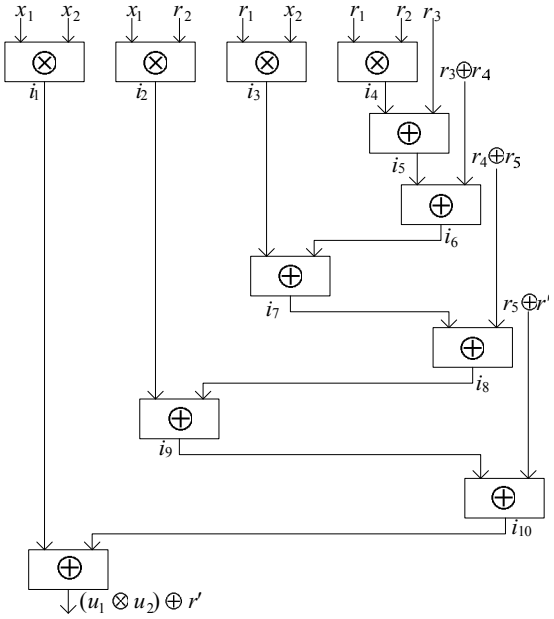

Figure 4: The Masked Multiplication

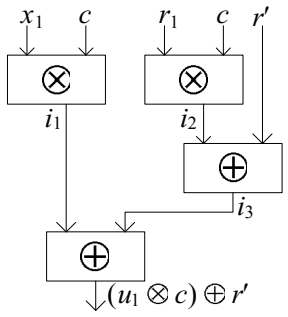

Figure 5: The Masked Mulc

The key schedule in AES algorithm can also be implemented as the sequence of masked primitive operations. And the transformations in inverse cipher including InvSubBytes and InvMixColumns can also decomposed to masked primitive operations. For space saving, their implementations are not included in this paper.

4. ARCHITECTURE OF THE AES COPROCESSOR

All the transformations in AES can be implemented using the mentioned masked primitive operations. So the data-path of one round in AES is equivalent to a graph in which each node represents a masked primitive operation. And each byte of the State is processed and updated in parallel in our coprocessor. The block diagram of the arithmetic core to perform the round function is shown in Fig. 6. The core may be a pipeline stage in an entire AES coprocessor, or it is the only unit to perform each round function iteratively.

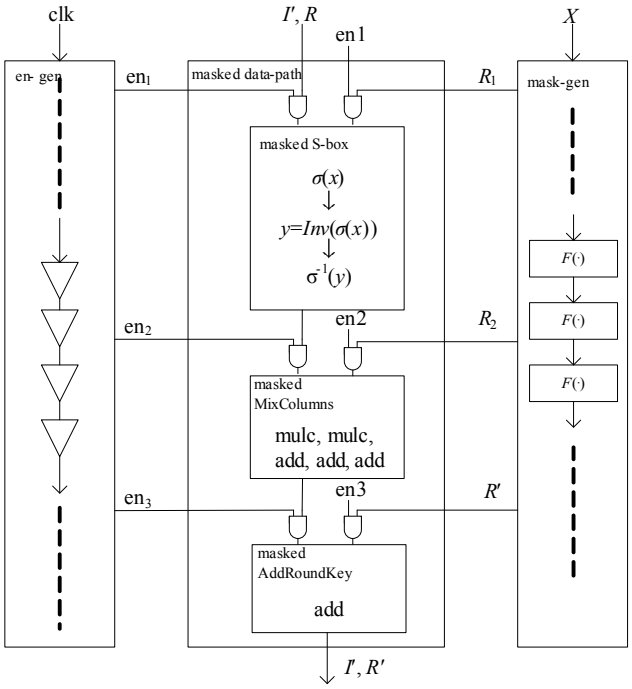

Figure 6: The Architecture of the Arithmetic Core

Let I' be the masked State and R be the corresponding mask, i.e. $I' = I \oplus R$, where I is the State. The arithmetic core takes I' and R as the input and generates new masked State I' which is masked by the newly generated mask R'.

For simplicity, we do not describe the practical implementation of the masked transformations including S-box, MixColumns and AddRoundKey in details. The key to implement the arithmetic core has two sides. The first one is to avoid the side-channel leakage caused by different arrival time of the input signals of the masked circuits [21]. This is achieved by the sub-block 'en-gen'. And the second one is to generate the required random masks by the sub-block 'mask-gen'.

(1) The Sub-block en-gen

To ensure nearly the same arrival time of all input signals of a masked primitive circuit, the input signals together with the generated enable signal are transmitted to the masked circuit through AND gates. The enable signals are generated by the sub-block en-gen using sequential buffers as shown in Fig. 6. When the enable signal is 1, the input signals to the masked circuit are valid. Otherwise, the input signals are zero. In other words, the input signals are synchronized by the enable signal. The sub-block en-gen takes the clock signal as input. So clk passes through the buffers one by one. In one clock cycle, 1 and 0 are transmitted alternately.

Let the critical path consists of the following primitive operations, $P_1, P_2, \ldots, P_n$ ($n > 1$). The timing constraint can be defined as

$$t_r + t_d > m_0 \times t_d \geq t_r$$
$$\max(t_r, t_i) + t_d > m_i \times t_d \geq \max(t_r, t_i),\ n > i \geq 1$$
$$T \geq (m_0 + m_1 + \ldots + m_{n-1}) \times t_d + t_n \geq t_r + t_1 + \ldots + t_n$$

where m_i ($n > i \geq 0$) is the number of buffer stages for the primitive operation P_i, t_r is the delay to generate random mask, t_d is the delay of a buffer. And t_i ($n \geq i \geq 1$) is the delay of the primitive operation P_i, T is the clock cycle. The waveform of the enable signals is shown in Fig. 7.

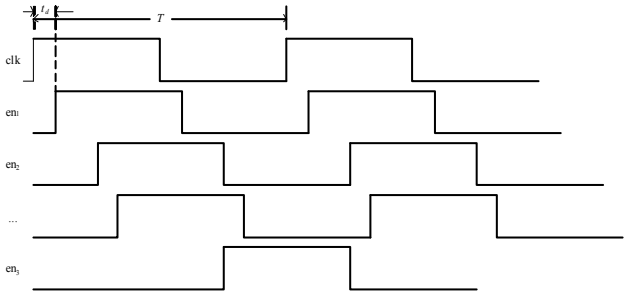

Figure 7: The Waveform of Enable Signals

The difference of the arrival time of two different input signals must not exceed $T/2$, where T is the clock cycle. Otherwise, several buffers can be added for the input signals which are generated much earlier. So that the arrival time of different inputs can be balanced. For example, we can add one buffer stage for the input x_2 of the masked addition when the difference of the arrival time of x_1 and x_2 exceeds $T/2$.

(2) The Sub-block Mask-gen

The sub-block mask-gen generates all the required random masks to mask each intermediate variable. It takes X as the seed, where X is the truly random value generated outside the AES coprocessor and updated every clock. In other words, X is uniformly distributed. The implementation of mask-gen is shown in Fig. 8 in details.

P1, P2, …, and Ps are permutations that change the position of each bit in X. P1(X), P2(X), …, Ps(X) are also uniformly distributed. The key in mask-gen is the function F which generates an m-bit random mask using an m-bit input. And F generates an m-bit random mask iteratively. Like the LFSR based random sequence generation, an irreducible polynomial with degree m is introduced and defined as

$$f(x) = x^m + a_{n-1}x^{m-1} + \ldots + a_1 x + 1, a_i \in \{0,1\}, 0 < i < m$$

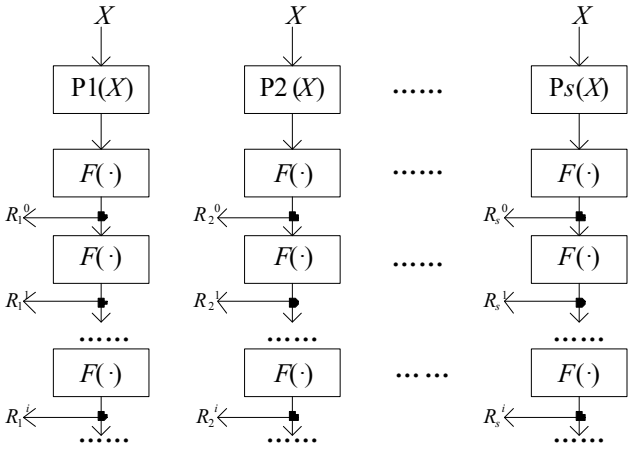

Figure 8: The Implementation of Mask-gen

Let the input of F be $(b_{m-1}, \ldots, b_1, b_0)$. The generated random mask $(b_{m-1}', \ldots, b_1', b_0')$ is defined as

$$b_0' = b_0 \wedge (a_1 b_1) \wedge \ldots \wedge (a_{m-1} b_{n-1})$$
$$b_1' = b_1 \wedge (a_1 b_2) \wedge \ldots \wedge (a_{m-1}\ b_0')$$
$$\ldots\ldots$$
$$B_{m-1}' = b_{n-1} \wedge (a_1\ b_0') \wedge \ldots \wedge (a_{m-1}\ b_{n-2}')$$

If the coefficient a_i ($1 \leq i \leq m$-1) of $f(x)$ is zero, the corresponding terms containing a_i to compute b_j' ($0 \leq j \leq m$-1) are not necessary.

To simplify the hardware complexity, we use the polynomial $f(x) = x^{16} + x + 1$, i.e., the function F generates a 16-bit random mask every time. And X is a 16-bit random value too. Then the output of F is defined as, $b_0' = b_0 \wedge b_1$, $b_1' = b_1 \wedge b_2$, …, $b_{15}' = b_{15} \wedge b_0' = b_{15} \wedge (b_0 \wedge b_1)$. The delay to generate one mask is equivalent to the delay of 2 XOR operations (to compute b_{15}'). So we can conclude that $t_r < t_i$ ($1 \leq i \leq n$-1). To mask 16 bytes of the State, 8 set of generators are placed in mask-gen, i.e., $s = 8$. The number of stages of generating function F is equal to the number of required random masks in the masked data-path.

In addition, the generated masks are alternately used. For a certain byte I_{ij} ($0 \leq i, j \leq 3$) (the byte in i-th row and j-th column), all the primitive operations $P_1, P_2, \ldots, P_n$, in the data-path of I_{ij} use the masks from different generator alternately. For example, the data-path of I_{00} use the following random masks $R_1^0, R_2^1, R_3^2, \ldots, R_8^7, R_1^8, \ldots$, and so on.

5. SECURITY AND PERFORMANCE

5.1 Security Analysis

To demonstrate the security of our implementation, we have to determine all the intermediate variables do not satisfy the precondition of GDPA.

Lemma 1. The function F generates an m-bit uniformly distributed mask b while its input a ($b = F(a)$) is a uniformly distributed random value. And the average Hamming distance of b and a is $m/2$.

The proof of this lemma is omitted. The generated masks are alternately used. And there exists weak correlation of two masks from different generators. Their average Hamming distance is $m/2$ too. So the weak correlation can not be used to improve the power analysis attack. It is considered that the masks for the data-path of a certain byte are nearly independent and uniformly distributed.

Lemma 2. Let $u \in \mathrm{GF}(2^8)$ be arbitrary, and c be a non-zero constant element of $\mathrm{GF}(2^8)$. Let r be uniformly distributed over $\{0, \ldots, 255\}$ independent of u. Then the variables, $u \oplus r$, $c \otimes (u \oplus r)$, $f_{(L,c)}(u \oplus r)$ and r^2 are uniformly distributed.

Lemma 3. Let $u_1, u_2 \in \mathrm{GF}(2^8)$ be arbitrary, and r_1, r_2 be independently and uniformly distributed over $\{0, \ldots, 255\}$. Set $I_1 = u_1 \oplus r_1$ and $I_2 = u_2 \oplus r_2$. Then the distributions of $I_1 \otimes I_2$, $r_1 \otimes I_2$, $I_1 \otimes r_2$, and $r_1 \otimes r_2$ are independent of u_1 and u_2 and identical to the distribution of a random variable z in $\mathrm{GF}(2^8)$. The distribution of z is defined as

$$\Pr(z = i) = \begin{cases} 511/2^{16}, i = 0 \\ 255/2^{16}, \text{otherwise} \end{cases} \quad (3)$$

The proofs of these two lemmas are straightforward and therefore omitted. Now we will show the security of the masked primitive operations.

In the masked addition, i_1, i_2, i_3, and the result $(u_1 \oplus u_2) \oplus r'$ are uniformly distributed according to lemma 2. In the masked affine transformation, i_1, i_2, and the result $f_{(L,c)}(u_1) \oplus r'$ are also uniformly distributed. In the masked square, the distributions of i_1 and i_2 are identical to the distribution shown in Eq. (3) according to lemma 3. And i_3 and the result $u_1^2 \oplus r'$ are uniformly distributed. In the masked constant multiplication, i_1, i_2, i_3, and the result $(c \otimes u_1) \oplus r'$ are uniformly distributed too. In the masked multiplication, the distributions of i_1, i_2, i_3 and i_4 are identical to the one in Eq. (3). And i_5, i_6, i_7, i_8, i_9, i_{10} and the result $(u_1 \otimes u_2) \oplus r'$ are all uniformly distributed.

According to the above analysis, we can see that the distribution of each intermediated variable in our implementation is independent of the sensitive data (i.e. key). In other word, there is no single intermediate variable that satisfies the precondition of GDPA. So our AES implementation is secure against first order DPA attack. Now we will discuss the security against high order DPA attacks of our implementation.

Lemma 4. Let $x_1, x_2, \ldots, x_n$ ($n > 0$) be n masked values, where $x_i = u_i \oplus r_i$, $0 \le i \le n$, and $r_1, \ldots, r_n$ are independently and uniformly distributed random masks. The joint distribution of the n masked values is independent of $u_1, u_2, \ldots, u_n$ so that these n variables do not satisfy the precondition of GDPA.

Since the n random masks are independently and uniformly distributed, the n masked values are independently and uniformly distributed too. The proof of this lemma is omitted here.

In the masked primitive operations, we can conclude that the joint distributions of multiple intermediate results calculated at different time are independent of the key and plaintext. For instance, in the masked multiplication, the distributions of the pairs of (i_p, i_q) $(1 \le p \le 4, 5 \le q \le 10)$, (i_6, i_7), (i_8, i_9), and (i_9, i_{10}) are independent of u_1 and u_2. According to lemma 4, the other pairs of variables are independent of u_1 and u_2 too. The four variables, i_1, i_2, i_3, and i_4 are produced at nearly the same time, so the attack on (i_1, i_2, i_3, i_4) is not feasible. The other combination of more than 2 variables can be analyzed similarly.

Since all the input signals of a masked primitive operation are synchronized, the side-channel leakage pointed out in [21] caused by different arrival time is avoided. To sum up, the proposed AES implementation is secure against the first order and high order DPA attacks.

To validate the security of the proposed scheme, we have performed a first order DPA attack on the practical implementation. This attack is mounted on the output of S-box in the first round, i.e. $S(M_{00} \oplus K_{00})$, where M and K are the plaintext and key respectively, and M_{00}, K_{00} denote the first byte of M and K. Here K_{00} is the target to break. And the correlation power analysis is used so that we calculate the correlation coefficient between the power traces and the Hamming Weight of $S(M_{00} K_{00})$. Let the correct value of K_{00} be 0x9C. The result of the attack is shown in Fig. 9. The result for 1000 power traces is shown in the top, and the

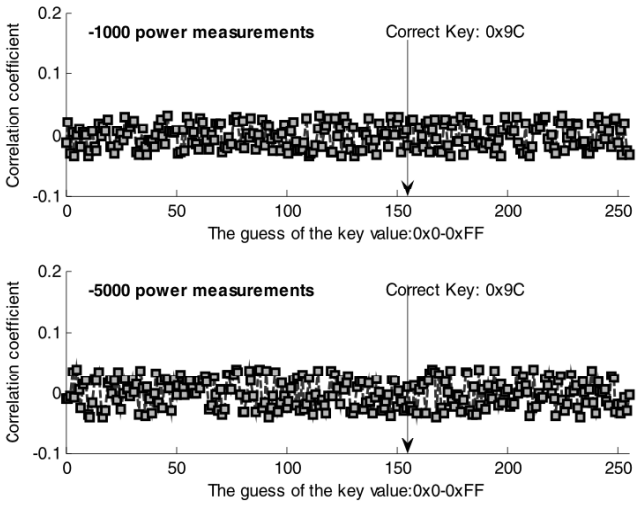

Figure 9: The Result of a Practical Attack

result for 5000 traces is shown in the bottom. We can see that both attacks fail to determine the correct key and it is useless to increase the number of power traces.

5.2 Performance Analysis

We have implemented an AES coprocessor which consists of only one arithmetic core presented in section 4. The coprocessor performs encryption/decryption with 128-bit key in 11 clock cycles. For comparison, we also implemented a common AES coprocessor in which no countermeasure is applied. And the S-box in the common coprocessor is implemented as a look-up table. We call the common coprocessor reference design. Using 0.18ìm CMOS technology, the experiment results of the two coprocessors under typical condition are shown in Table 1.

We can see that our design achieves resistance to high order DPA attacks at the cost of 3.1 times larger areas and 1.2 times greater delay than the common AES coprocessor.

Table 1
The Experiment Results of the Two Coprocessors

	Area (gates)	*Clock Freq. (MHz)*	*Throughput (Mbps)*	*Throughput/ Area(Kbps/gates)*
this paper	140K	180	1100	7.9
reference design	45K	220	1342	29.8

6. CONCLUSION

In this paper, we have proposed an AES implementation secure against high order DPA attacks based on masking scheme. Several masked primitive operations are defined and all the transformations in AES are then transformed to the sequence of these primitive operations. The composite field arithmetic over $GF(((2^2)^2)^2)$ is used to implement inversion efficiently. The arithmetic core of the AES coprocessor is presented. Besides the masked data-path, two additional sub-blocks are introduced. The one is to generate enable signals to synchronized all the input signals of masked primitive operations. And the other one is to generate the required random values to mask all the intermediate variables. Theoretical analysis shows that our implementation is secure against first order and high order DPA attacks.

In the future works, we will try to reduce the required hardware cost and computation without loss of security.

ACKNOWLEDGMENTS

The authors would like to thank the Natural Science Foundation of China (NSFC) for funding this research project (No. 60706026).

REFERENCES

[1] P. C. Kocher, B. Jun, “Differential Power Analysis”, in Advances in Cryptology, 1999, LNCS **1666**, 388-397.

[2] T. Messerges, E. A. Dabbish, R. H. Sloan, “Examining Smart-Card Security under the Threat of Power Analysis Attacks”, *IEEE Transaction on Computers*, **51**(5), 2002, 541-552.

[3] M. Joye, P. Paillier, B. Schoenmakers, “On Second-Order Differential Power Analysis”, in Cryptographic Hardware and Embedded Systems, 2005, LNCS **3659**, 293-308.

[4] E. Brier, C. Clavier, F. Olivier, “Correlation Power Analysis with a Leakage Model”, in Cryptographic Hardware and Embedded Systems, 2004, LNCS **3156**, 16-29.

[5] M. L. Akkar, C. Giraud, “An Implementation of DES and AES, Secure against Some Attacks”, in Cryptographic Hardware and Embedded Systems, 2001, LNCS **2162**, 309-318.

[6] Y. J. Baek, M. J. Noh, “DPA-Resistant Finite Field Multipliers and Secure AES Design”, in ISPEC, 2006, LNCS **3903**, 1-12.

[7] J. Blömer, V. Krummel, “Provably Secure Masking of AES”, in Selected Areas in Cryptography, 2005, LNCS **3357**, 60-83.

[8] N. T. Courtois, L. Goubin, “An Algebraic Masking Method to Protect AES Against Power Attacks”, in Information Security and Cryptology, 2006, LNCS, **3935**, 199-209.

[9] J.D. Goliæ, “Multiplicative Masking and Power Analysis of AES”, in Cryptographic Hardware and Embedded Systems, 2003, LNCS **2523**, 198-212.

[10] C. Herbst, E. Oswald, S. Mangard, “An AES Smart Card Implementation Resistant to Power Analysis Attacks”, in ACNS, 2006, LNCS **3989**, 239-252.

[11] E. Oswald, S. Mangard, N. Pramstaller, V. Rijmen, “A Side-Channel Analysis Resistant Description of the AES S-box”, in Fast Software Encryption, 2005, LNCS **3557**, 413-423.

[12] E. Oswald, K. Schramm, “An Efficient Masking Scheme for AES Software Implementations”, in WISA, 2005, LNCS **3786**, 292-305.

[13] E. Prouff, C. Giraud, S. Aumônier, “Provably Secure S-Box Implementation Based on Fourier Transform”, in Cryptographic Hardware and Embedded Systems, 2006, LNCS **4249**, 216-230.

[14] A. G. Rostovtsev, O. V. Shemyakina, “AES Side Channel Attack Protection using Random Isomorphisms”, 2005, Available on *http://eprint.iacr.org/2005/087.pdf*.

[15] K. Schramm, C. Paar, “Higher Order Masking of the AES”, in CT-RSA, 2006, LNCS **3860**, 208-225.

[16] E. Trichina, L. Korkishko, “Secure and Efficient AES Software Implementation for Smart Cards”, in WISA, 2004, LNCS **3325**, 425-439.

[17] E. Trichina, D. D. Seta, L. Germani, “Simplified Adaptive Multiplicative Masking for AES”, in Cryptographic

Hardware and Embedded Systems, 2003, LNCS **2535**, 187-197.

[18] T. Popp, S. Mangard, "Masked Dual-Rail Pre-charge Logic: DPA-Resistance Without Routing Constraints", in Cryptographic Hardware and Embedded Systems, 2005, LNCS **3659**, 172-186.

[19] K. Tiri, I. Verbauwhede, "Securing Encryption Algorithms against DPA at the Logic Level: Next Generation Smart Card Technology", in Cryptographic Hardware and Embedded Systems, 2003, LNCS **2779**, 137-151.

[20] Y. Tong, Z. Wang, K. Dai, H. Lu, "Designing Power Analysis Resistant and High Performance Block Cipher Coprocessor Using WDDL and Wave-Pipelining", in Inscrypt, 2006, LNCS **4318**, 66-77.

[21] S. Mangard, K. Schramm, "Pinpointing the Side-Channel Leakage of Masked AES Hardware Implementations", in Cryptographic Hardware and Embedded Systems, 2006, LNCS **4249**, 76-90.

[22] M. L. Akkar, R. Bevan, L. Goubin, "Two Power Analysis Attacks against One-Mask Methods", in FSE, 2004, LNCS **3017**, 332-347.

[23] S. Chari, C. S. Jutla, J. R. Rao, R. Rohatgi, "Towards Sound Approaches to Counteract Power-Analysis Attacks", in Crypto, 1999, LNCS **1666**, 398-412.

[24] J. S. Coron, E. Prouff, M. Rivain, "Side Channel Cryptanalysis of a Higher Order Masking Scheme", in Cryptographic Hardware and Embedded Systems, 2007, LNCS **4727**, 28-44.