IET Information Security

Research Article

New second-order threshold implementation of AES

ISSN 1751-8709
Received on 27th May 2018
Revised 9th November 2018
Accepted on 20th November 2018
E-First on 29th January 2019
doi: 10.1049/iet-ifs.2018.5244
www.ietdl.org

Yongzhuang Wei[1,2] ✉, *Fu Yao*[2], *Enes Pasalic*[1,3], *An Wang*[4]

[1]*Guangxi Key Laboratory of Cryptography and Information Security, Guilin University of Electronic Technology, Guilin, People's Republic of China*
[2]*Guangxi Colleges and Universities Key Laboratory of Cloud Computing and Complex Systems, Guilin University of Electronic Technology, Guilin, People's Republic of China*
[3]*University of Primorska, FAMNIT, Koper, Slovenia*
[4]*School of Computer Science, Beijing Institute of Technology, Beijing, People's Republic of China*
✉ *E-mail: walker_wyz@guet.edu.cn*

Abstract: In this work, the authors propose some alternative hardware efficient masking schemes dedicated to protect the Advanced Encryption Standard (AES) against higher order differential power analysis (DPA). In general, the existing masking schemes all have in common an intrinsic trade-off between the two main parameters of interest, namely the generation of fresh random masking values and the cost of hardware implementation. The design of efficient masking schemes which are non-expensive in both aspects appears to be a difficult task. In this study, the authors propose a second-order threshold implementation of AES, which is characterised by a beneficial trade-off between the two parameters. More precisely, compared to the masking scheme of De Cnudde *et al.* at CHES 2016, which currently attains the best practical trade-off, the proposed masking scheme requires 28.4% less random masking bits, whereas the implementation cost is slightly increased for about 13.7% (thus the chip area is 1.4 kGE larger). This masking scheme has been used to implement AES on an field-programmable gate array (FPGA) platform and its resistance against the second-order DPA in a simulated attack environment has been confirmed.

1 Introduction

Time and power consumption along with electromagnetic radiation of an encryption algorithm are the main sources of the information leakage which leads to powerful cryptanalytic attacks known as side-channel analysis (SCA) [1]. In general, embedded systems that employ cryptographic schemes must be protected against SCA and these countermeasures (implemented either in hardware or software) are commonly referred to as masking schemes. One of the most popular techniques of SCA is differential power analysis (DPA) [2] whose core idea is to use the statistical model to analyse the power consumption (power traces) collected from the physical implementation of the encryption algorithm. Then, based on the knowledge of hardware implementation of the considered cryptographic algorithm, the attacker can efficiently recover (a portion of) the secret key.

The Rijndael encryption algorithm [3] was selected as Advanced Encryption Standard (AES) in 2001, and due to its worldwide use, its protection against SCA is of essential importance. In order to resist DPA attacks, a natural countermeasure on circuit level is to balance the power consumption of the algorithm so that the leakage of the information is minimised. On the other hand, for algorithmic level protection, the main idea is to keep the intermediate values (computation at different rounds) of the cryptographic algorithms and the key information independent [4]. The most common practice is to employ the so-called masking, which splits each sensitive intermediate variable of the cryptographic algorithm into multiple shares (using the ideas of secret sharing), and then performs the computation on these shares instead. These shares of the sensitive intermediate variables are never combined in a way so that any sensitive variable is unmasked during the entire encryption process.

Whereas the masking of linear operations is relatively easy, the same process is more subtle when the so-called substitution boxes (S-boxes) are concerned (commonly the only source of non-linearity in block ciphers). If $x \in GF(2^n)$ is the sensitive variable that needs to be masked, then we consider the equation $x_1 \oplus \ldots \oplus x_s = x$, $x_i \in GF(2^n)$, where each x_i is called a masking share $(i = 1, \ldots, s)$. In particular, for any given S-box which performs a non-linear transformation of the input, the masking shares of both input and output are selected in such a way that the operations performed on the input masking shares give a correct output when the output masking shares are recombined. These masking variables propagate throughout the cipher in such a manner that every intermediate variable is independent of any sensitive variable. This strategy ensures that the instantaneous leakage is independent of any sensitive variable, thus rendering SCA difficult to perform. The masking process can be characterised by the number of random masks used per sensitive variable and basically a dth-order masking, involving d masks (random numbers), can be theoretically broken by a $(d+1)$th-order SCA [5, 6].

Many different masking schemes have been developed during the past two decades [7–9]. However, it was shown [10–12] that masking can still be vulnerable to a first-order DPA due to the presence of glitches which are typical for certain hardware implementations. Therefore, the masking schemes that use so-called threshold implementation (TI) have been developed in order to provide the resistance against DPA even in the presence of glitches [13]. A more detailed overview of the existing masking schemes, particularly those that employ the TI design rationales, is given in the next subsection. We emphasise that in this article we entirely focus on efficient masking schemes of TI type dedicated to the protection of the Rijndael algorithm, thus only briefly mentioning other masking techniques.

1.1 Related work

Apart from other masking methods [14–16], during the last decade many attempts have been made towards an efficient protection of the Rijndael algorithm using TI-based masking. In particular, Bilgin *et al.* [17] presented an optimised implementation of a first-

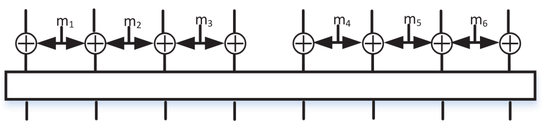

Fig. 1 *Structure of (re)masking*

order masking scheme of AES that uses 44 random bits per round, whose total implementation cost is ∼9 kGE (kilo gate equivalent). To handle higher order DPA attacks, a second-order masking scheme of the AES-128 (the key length being 128 bits) was proposed in [18]. This gave a rather significant increase of both relevant parameters, requiring 126 random bits per encryption round and the total chip area of 18.6 kGE. In 2016, De Cnudde *et al.* [19] introduced a general method to protect AES-128 against dth-order DPA attacks using $d+1$ masking shares. In particular, to implement the second-order masking of AES-128, this approach uses 162 random bits per round and the cost of hardware implementation is around 10 kGE. This is currently the most efficient implementation of a second-order TI-based masking of AES-128 in terms of the trade-off between the hardware cost and the amount of random bits used. We notice that a second-order domain-oriented masking of AES-128 was proposed in [20], which requires only 84 random masking bits and having the total chip area of about 12.8 kGE. However, this countermeasure is conducted on circuit level rather than providing the algorithmic level protection.

1.2 Our contribution

The optimisation of the two most relevant parameters (the amount of random bits used and hardware cost) for TI masking schemes, or alternatively providing a scheme with a satisfactory trade-off between the parameters, is a challenging task. In this article, we propose a new second-order TI masking scheme dedicated to AES-128 which is essentially derived from a more general framework for constructing $(d+1)$th-order masking scheme by efficiently employing the existing ones of order d. Thus, to design a second-order TI for AES, we will employ a suitable first-order masking scheme. In general, to build a $(d+1)$th-order masking scheme, our method uses the relations of a given dth-order masking scheme and split these if necessary to build new relations which ensure the resistance to $(d+1)$th-order DPA attacks. These relations however satisfy the three basic properties of TI-based schemes, namely correctness, non-completeness and uniformity. The advantage of such an approach is that a higher-order masking scheme can be constructed without increasing the number of input masking shares, which then efficiently reduces the cost of hardware implementation. In addition, the design of such a masking scheme only requires rather simple manual manipulation of the existing relations and no automated search techniques are needed.

Our second-order TI masking scheme of AES-128 is implemented on the FPGA platform and its resource consumption is evaluated using the NanGate 45 nm Open Cell Library [21]. It is shown that our method requires only 116 random bits to implement masking in each encryption round. In addition, the hardware implementation of the S-box module requires 0.43 kGE and a total chip area of hardware implementation is about 11.9 kGE. The amount of random bits used in our scheme is therefore 28.4% smaller compared to the currently most efficient implementation of a second-order TI masking scheme proposed by De Cnudde *et al.* [19], though the chip area is slightly increased by ∼13.7% (see also Section 5.2). In more detail, to make the masking expressions uniform some fresh random values are added as depicted in Fig. 1. This approach implies a reduction of their size compared to the structure of masking scheme in [19]. On the other hand, our method of masking S-boxes requires four masking shares which is one more than that of the scheme in [19], and therefore the total chip area is slightly increased. The security of our masking scheme of AES-128 is verified against second-order DPA attacks in an actual attack scenario and the implementation is confirmed to be secure even under the assumption that the attacker is capable of collecting a huge amount of power traces.

The rest of the paper is organised as follows. In Section 2, a brief description of AES and TI is given. A new second-order masking scheme for AES-128 is presented in Section 3. In Section 4, a detailed implementation of the masking scheme of AES-128 on FPGA platform is discussed. The resistance of AES-128 against second-order DPA in a simulated attack environment is evaluated in Section 5. Some concluding remarks are given in Section 6.

2 Preliminaries

We here briefly recall the structure of AES and introduce the basic concepts related to TI.

2.1 Brief description of advanced encryption standard

AES uses a 128-bit data block length, and a variable length secret key of sizes 128, 192 or 256 bits (usually denoted by AES-128, AES-192, AES-256, respectively). In particular, for AES-128, AES-192 and AES-256, the corresponding number of iterative round functions is 10, 12 and 14, respectively. For the Rijndael family, if the data block length is 256-bit, then the length of secret key is of the same fixed size as the block length. In this case, the corresponding number of iterative rounds is 14.

Prior to the encryption process, the plaintext and the secret key are arranged in the so-called state matrix, which for the standard sizes of block length and key length being 128 bits is a quadratic four times four matrix $\boldsymbol{A} = (a_{i,j})$, $i, j = 0, \ldots, 3$, whose entries are bytes. The same notation is used if we refer to the round key matrix, though the entries are replaced by $K_{i,j}$ to indicate that we refer to key bytes. In each round, the four operations AddRoundKey, SubBytes, ShiftRows and MixColumns (see Fig. 2) are performed which are specified as follows.

i. *AddRoundKey*: The elements of the state matrix are directly XORed with the bytes corresponding to a round key (derived from the master secret key).
ii. *SubBytes*: SubBytes is the only non-linear component of the AES encryption algorithm. This operation acts on each byte a of the state matrix $\boldsymbol{A}$ so that its multiplicative inverse a' in $GF(2^8)$ is computed (by convention if $a=0$ then $a'=0$). Then, representing $a' = (a_0', a_1', \ldots, a_7')$, where $a_i' \in GF(2)$, the affine transformation over $GF(2)^8$ is performed

$$\begin{pmatrix} b_0\\ b_1\\ b_2\\ b_3\\ b_4\\ b_5\\ b_6\\ b_7 \end{pmatrix} = \begin{pmatrix} 1&1&1&1&1&0&0&0\\ 0&1&1&1&1&1&0&0\\ 0&0&1&1&1&1&1&0\\ 0&0&0&1&1&1&1&1\\ 1&0&0&0&1&1&1&1\\ 1&1&0&0&0&1&1&1\\ 1&1&1&0&0&0&1&1\\ 1&1&1&1&0&0&0&1 \end{pmatrix} \begin{pmatrix} a_0'\\ a_1'\\ a_2'\\ a_3'\\ a_4'\\ a_5'\\ a_6'\\ a_7' \end{pmatrix} \oplus \begin{pmatrix} 0\\ 1\\ 1\\ 0\\ 0\\ 0\\ 1\\ 1 \end{pmatrix} \tag{1}$$

which provides a new state matrix whose entries are $(b_0, \ldots, b_7) \in GF(2)^8$.
iii. *ShiftRows*: The rows of the state matrix are shifted cyclically to the left leaving the first row unchanged. The elements of the second, third and fourth row are shifted by one, two and three positions to the left, respectively.
iv. *MixColumns*: This operation can be represented as a simple multiplication of the state matrix with the constant matrix $\boldsymbol{M}$ (see [3] for further details)

$$\boldsymbol{M} = \begin{pmatrix} 02&03&01&01\\ 01&02&03&01\\ 01&01&02&03\\ 03&01&01&02 \end{pmatrix}. \tag{2}$$

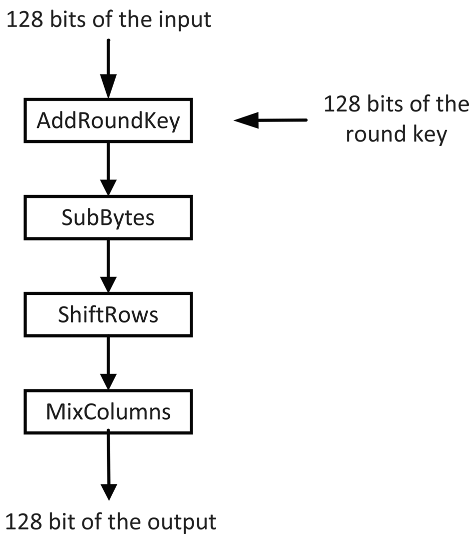

Fig. 2 *Round function of the AES*

2.2 Threshold implementation

The first TI was proposed by Nikova *et al.* in 2006 [13], combining secret sharing, threshold cryptography and multi-party computation. The main advantage of TI methods, compared to some other masking schemes, is their ability to resist attacks based on glitches apart from providing security against DPA cryptanalysis (of certain order). In addition, the random masks are only added to the input variables and the encryption algorithm is executed on these masking shares without any further transforms between the sensitive variables and masking shares. Some useful notations are given below.

- $GF(2^n)$ denotes the finite field of order 2^n, which is isomorphic to the vector space $GF(2)^n$.
- $x \in GF(2^n)$ denotes the sensitive variable that needs to be masked.
- $x_i \in GF(2^n)$, for $i = 1, \dots, s$, denotes the masking shares of the variable x.
- $\overline{\boldsymbol{x}} = (x_1, \dots, x_{s_{in}}) \in GF(2^n)^{s_{in}}$ contains the masking shares of the variables x.
- $s_{\text{in/out}}$ denotes the number of the input/output masking shares.
- $\oplus$ denotes the Boolean componentwise XOR operation.
- $Sh(x)$ denotes the set of all available masking vectors $\overline{\boldsymbol{x}}$ for the variable $x \in GF(2)^n$, that is, $Sh(x) = \{\overline{\boldsymbol{x}} = (x_1, \dots, x_s) \in GF(2)^{ns} | x_1 \oplus \dots \oplus x_s = x\}$.

The main objective of TI masking schemes is to efficiently protect both non-linear and linear functions of a given block cipher. In what follows, we give a brief overview of some common rationales when TI masking is applied for protection of (non)linear functions.

2.2.1 TI for linear functions: Assume that a given linear function is of the form $z = L(x, y)$, where $x, y, z \in GF(2^n)$. To resist dth-order DPA, the masking expression needs to be constructed using the following three steps:

i. *Decomposition of input variables* x, y: Since the algebraic degree of any linear function is one, the number of either input or output masking shares is $s_{in} = s_{out} = d + 1$, which is sufficient to resist dth-order DPA attacks. The input masking shares are calculated as $x_{s_{in}} = \oplus_{i=1}^{s_{in}-1} x_i \oplus x$ and $y_{s_{in}} = \oplus_{i=1}^{s_{in}-1} y_i \oplus y$, where $x_1, \dots, x_{s_{in}-1}, y_1, \dots, y_{s_{in}-1} \in GF(2^n)$ are masking shares selected randomly from a uniform distribution.

ii. *Linear function is expressed by a masking vector*: The input variables x, y in the linear function $z = L(x, y)$ are replaced by the masking vectors $\overline{\boldsymbol{x}} = (x_1, \dots, x_{s_{in}}), \overline{\boldsymbol{y}} = (y_1, \dots, y_{s_{in}})$ so that $z = L(\oplus_{i=1}^{s_{in}} x_i, \oplus_{i=1}^{s_{in}} y_i)$.

iii. *Decomposition of a linear function*: Let $z_i = L(x_i, y_i) \in GF(2^n)$, for $i = 1, \dots, s_{in}$, be the output masking shares specified for different input tuples (x_i, y_i). Then, due to linearity of L, we have

$$z = \bigoplus_{i=1}^{s_{in}} z_i = \bigoplus_{i=1}^{s_{in}} L(x_i, y_i) = L(\bigoplus_{i=1}^{s_{in}} x_i, \bigoplus_{i=1}^{s_{in}} y_i) = L(x, y). \tag{3}$$

The above method efficiently provides masking schemes against dth-order DPA for a given linear function L. Since there is only one input masking share in each component expression $z_i = L(x_i, y_i)$, the input and output values $z = L(x, y)$ cannot be compromised even if the attacker probes any d wires of the circuits [22] due to the condition $s_{\text{in}} = s_{\text{out}} = d + 1$.

2.2.2 TI for non-linear function: Similarly to the process of the chapter 2.2.1, we assume that the expression for a given non-linear function is $z = N(x, y)$, $x, y, z \in GF(2^n)$. The masking expression of a non-linear function z, that resists dth-order DPA, can be obtained using in a similar manner as above. The decomposition of input variables x, y, along with their replacement by the masking vectors $\overline{\boldsymbol{x}} = (x_1, \dots, x_{s_{in}}), \overline{\boldsymbol{y}} = (y_1, \dots, y_{s_{in}})$, gives the representation $z = N(\oplus_{i=1}^{s_{in}} x_i, \oplus_{i=1}^{s_{in}} y_i)$.

Concerning the decomposition of a non-linear function $z = N(x, y)$, we notice the following. To resist dth-order DPA, the input and output values of $z = N(x, y)$ cannot be leaked, even if the attacker probes any d wires of the circuit [22]. Therefore, the number of the input masking shares contained in each masking expression cannot exceed $s_{in} - d$. On the other hand, when the terms of $z = N(\oplus_{i=1}^{s_{in}} x_i, \oplus_{i=1}^{s_{in}} y_i)$ are expanded they still need to be recombined to form the s_{out} masking shares which then need to satisfy the three basic requirements (that we list below) to ensure protection against dth-order DPA attacks.

Property 1: (Correctness): For each variable $x \in GF(2^n)$, if it is masked, then the Boolean XOR operation of all masking shares should be equal to x, that is, $\oplus_i x_i = x$. The same applies to the output masking shares, after the input has been processed by some (non)linear function f_i, so that the sum of output masking shares is z, i.e. $z = \oplus_i z_i = \oplus_i f_i(x)$.

Property 2: (dth-order non-completeness): In order to resist dth-order DPA, any combination of up to d component functions f_i is independent of at least one input masking share (where $f_1, \dots, f_s$ are shares of some function f). Therefore, the combined function cannot contain all the input masking shares.

Property 3: (Uniformity): For each input variable x and the output variable z, if their masking shares have the same probability distribution then the masking process is called a uniform masking.

If only linear functions are concerned, then the uniformity of the input masking shares can always ensure the uniformity of the output masking shares. However, for non-linear functions, in order to guarantee the uniformity of the output masking shares, we need to construct uniform function shares or even to add some additional random values to achieve uniformity.

3 New construction of second-order masking schemes

In this section, a new construction method of second-order masking schemes is proposed. The main idea behind our novel approach is a two-step design strategy. In the first step, we transform the original non-linear function $z = f(x, y) = xy$ ($x, y, z \in GF(2^n)$) to a new function that is resistant to first-order but not to second-order DPA attacks. In the second step, this function is further processed in terms of adding efficient masking schemes so that the resulting function achieves the resistance against the second-order DPA.

We emphasise that the overall process also pays attention to the cost of implementation in terms of the size of random numbers used as well as with respect to a total chip area.

Assume $z = f(x, y) = xy$, where $x, y, z \in GF(2^n)$. To resist first-order DPA, we transform this equation to (4) below by using Properties 1–3 given in Section 2.2.2, where $(x_1, \ldots, x_4, y_1, \ldots, y_4)$ and (z_1, z_2, z_3) are the input and output masking vectors, respectively. More precisely

$$\begin{aligned} z_1 &= x_2y_2 \oplus x_2y_3 \oplus x_3y_2 \oplus x_3y_3 \oplus x_4y_2 \oplus x_4y_3 \oplus y_4 \\ z_2 &= x_1y_1 \oplus x_1y_4 \oplus x_3y_1 \oplus x_3y_4 \oplus x_1y_3 \oplus x_4 \\ z_3 &= x_2y_1 \oplus x_1y_2 \oplus x_2y_4 \oplus x_4y_1 \oplus x_4y_4 \oplus x_4 \oplus y_4 \end{aligned} \tag{4}$$

We first transform the relationships given by (4) into

$$\begin{aligned} z_1 &= x_2y_2 \oplus x_2y_3 \oplus x_3y_2 \oplus x_3y_3 \\ z_2 &= x_1y_1 \oplus x_1y_3 \oplus x_3y_1 \\ z_3 &= x_1y_2 \oplus x_2y_1 \\ z_4 &= x_4y_4 \\ z_5 &= x_4y_2 \oplus x_4y_3 \\ z_6 &= x_1y_4 \oplus x_3y_4 \\ z_7 &= x_2y_4 \\ z_8 &= x_4y_1 \end{aligned} \tag{5}$$

to ensure that both correctness and the second-order non-completeness are then satisfied.

Moreover, in order to satisfy the uniformity property, we need to use some fresh random values in the output masking shares. Thus, (5) is transformed to

$$\begin{aligned} z_1 &= x_2y_2 \oplus x_2y_3 \oplus x_3y_2 \oplus x_3y_3 \oplus \boldsymbol{m}_1 \\ z_2 &= x_1y_1 \oplus x_1y_3 \oplus x_3y_1 \oplus \boldsymbol{m}_1 \oplus \boldsymbol{m}_2 \\ z_3 &= x_1y_2 \oplus x_2y_1 \oplus \boldsymbol{m}_2 \oplus \boldsymbol{m}_3 \\ z_4 &= x_4y_4 \oplus \boldsymbol{m}_3 \\ z_5 &= x_4y_2 \oplus x_4y_3 \oplus \boldsymbol{m}_4 \\ z_6 &= x_1y_4 \oplus x_3y_4 \oplus \boldsymbol{m}_4 \oplus \boldsymbol{m}_5 \\ z_7 &= x_2y_4 \oplus \boldsymbol{m}_5 \oplus \boldsymbol{m}_6 \\ z_8 &= x_4y_1 \oplus \boldsymbol{m}_6, \end{aligned} \tag{6}$$

where $\boldsymbol{m}_1, \ldots, \boldsymbol{m}_6$ denote these newly introduced random vectors.

A standard notation for specifying that k input and l output masking shares are used is (k, l), which in our case implies that (4, 8) shares are used (4 input shares for each input variable). The structure for the addition of random values m_i is depicted in Fig. 1, which is also used when performing remasking.

Remark 1: Note that if the number of the input masking shares is 5, one can achieve uniformity of the output masking share (expression) without additional random values. This approach was originally described as (5, 10) scheme in [23]. In detail, for one-bit AND operation, 16 AND gates, 8 XOR gates and 8 output registers are required for (4, 8) scheme. On the other hand, the (5, 10) scheme requires 25 AND gates, 20 XOR gates and 10 output registers, see [23]. In NanGate 45 nm Open Cell Library, 1 AND gate corresponds to 1.064 GE, 1 XOR gate equals 1.596 GE and a register requires 5.33 GE. Therefore, for a multiplication operation of one bit, the area of (5, 10) is 54.4% larger than the area for (4, 8) scheme but the former approach does not require six random masking values m_i.

3.1 Constructing $(d+1)$th-order masking schemes from dth-order masking schemes

The method given in the previous section can be easily generalised to yield a generic technique to construct $(d+1)$th-order masking schemes. The procedure can be described as follows:

i. Build a dth-order masking scheme using an 'optimal' method of TI. The number of masking shares for each input variable is denoted by S_{in}.
ii. Then, analyse the expressions for output masking shares: if the number of masking shares of each input variable contained in the expression is less than or equal to $S_{in} - d$, then such an expression needs not to be split; otherwise, split such an expression into new expressions until the new output masking shares contain at most $S_{in} - d$ input shares.
iii. Check whether the expression of each output masking shares satisfies the property of uniformity, if not, introduce fresh random values to satisfy it.

This procedure is very efficient when a low-order TI is considered and no sophisticated optimisation methods are needed assuming that the dth-order masking scheme is fixed and 'optimal'.

4 Our second-order TI of AES-128

In this section, based on the approach given in the previous section, we discuss the implementation of our second-order TI of AES-128 on the SAKURA-G platform which is an FPGA development board.

4.1 Analysis of the AES S-box

The algebraic degree of the inverse substitution boxes used in AES-128 is $t = 7$. Then to ensure the resistance against second-order DPA attacks, the number of required input masking shares is at least $s_{in} \geq t \times d + 1 = 7 \times 2 + 1 = 15$ if the method proposed in [13] is used. The cardinality of random values used for the protection of a single S-box is therefore quite large. On the other hand, similarly to the work in [24], the inverse operation over $GF(2^8)$ can be performed using the tower of fields $GF(2^8) > GF(2^4) > GF(2^2)$. This is quite helpful for reducing not only the algebraic degree but also the number of fresh random values. A detailed description of converting the inverse operation performed in $GF(2^m)$ to $GF(2^{m/2})$ is shown in Fig. 3. Notice that the inverse operation over $GF(2^2)$ corresponds to squaring, that is $x^{-1} = x^2$ in $GF(2^2)$, so that the only non-linear operation, when using the tower of fields $GF(2^2) < GF(2^4) < GF(2^8)$, is the multiplication in $GF(2^2)$ and $GF(2^4)$. Apparently, protecting the multiplication in $GF(2^2)$ and $GF(2^4)$ against a second-order DPA implies that the subbytes of the round function are also resistant against second-order DPA attacks.

Since the algebraic degree of multiplication $z = f(x, y) = xy$ in $GF(2^2)$ or $GF(2^4)$ is equal to two (considered as a multivariate equation), we can use (4, 8) as the masking scheme for the function $z = f(x, y) = xy$ to resist second-order DPA attacks. In particular, the use of registers, for storing the values obtained by each non-linear operation $z = f(x, y) = xy$, is necessary to prevent the propagation of glitches to the SubBytes modules.

4.2 Implementation of the round function of AES-128

A minimal requirement for ensuring the resistance of our hardware implementation against second-order DPA attacks implies the use of three, and respectively four, masking shares to protect the linear and non-linear part of a single round encryption process. A high-level overview of the main operations is depicted in Fig. 4.

Each round of the AES encryption process executes two different operations at the same time, namely the computation of

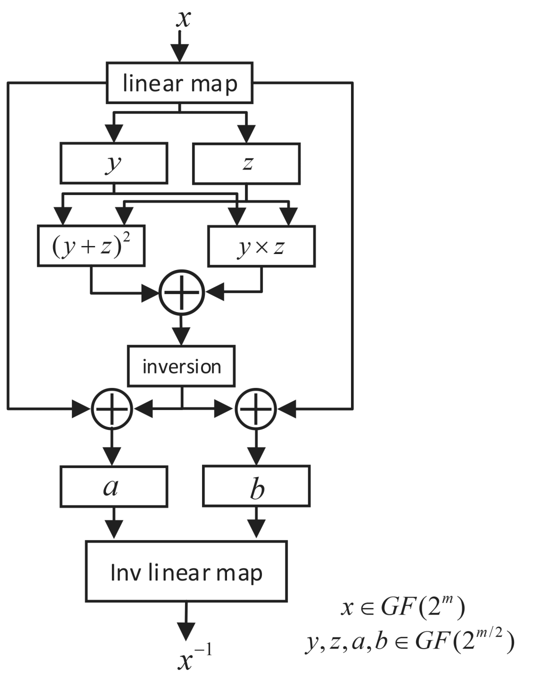

Fig. 3 *Converting the inverse operation from* $GF(2^m)$ *to* $GF(2^{m/2})$

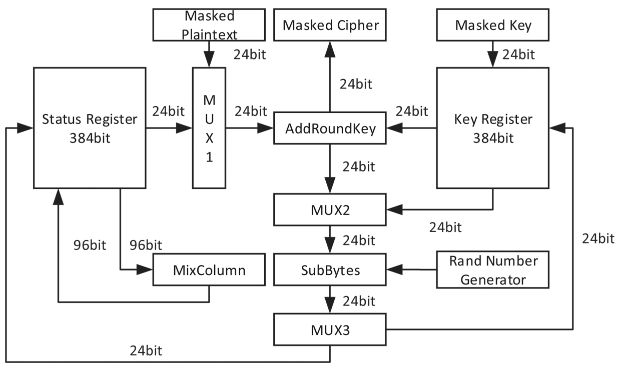

Fig. 4 *Hardware implementation of the second-order masking scheme*

the round function and the key expansion. In our implementation, the execution of a round function will be performed within 25 clock cycles. Within the round encryption process, the data is handled by the SubByte module from the 1st to the 20th clock cycle, which completes the application of the SubByte modules. The output data of the SubByte module is then processed during the 21st clock cycle to perform the operation ShiftRows. Finally, the output data of the operation ShiftRows is handled by the MixColumns module from the 22nd to the 25th clock cycle. On the other hand, the execution of the key expansion will be performed within 27 clock cycles. In more detail, the key bytes are stored during the first 16 cycles and the key expansion mechanism (that uses SubBytes module) is executed from the 17th to the 24th clock cycle. The update of the last three columns of the key state matrix is performed within the last three clock cycles, thus from 25th to 27th. The implementation of the main modules of the round encryption (which are SubBytes, Status register, Key register and Fresh random number generator module) is described as follows.

SubBytes module: The implementation of the SubBytes module uses a tower of fields approach mentioned earlier, where the selection of the bases and irreducible polynomials is the same as in [24, Sec. 2]. In detail, the defining polynomial of $GF(2^8)$ over $GF(2)$ is $x^8 + x^4 + x^3 + x + 1$; the basis of $GF(2^8)$ over $GF(2^4)$ is a normal basis, whose elements are the roots of $y^2 + y + \nu$ (which is irreducible over $GF(2^4)$ for certain $\nu \in GF(2^4)$); the basis of $GF(2^4)$ over $GF(2^2)$ is a normal basis, whose elements are the roots of $z^2 + z + N$ (irreducible over $GF(2^2)$ for certain $N \in GF(2^2)$). This module uses the pipeline technology and operates from the 1st to the 24th clock cycle. From the 1st to the 20th clock cycle, the data handled by this module is the data to be encrypted. From the 17th to the 24th clock cycle, this module processes the key bytes. It is worth noting that, due to the use of pipeline technology, this module handles two kinds of data simultaneously from the 17th to the 20th clock cycle; the data to be encrypted and the data related to the key expansion. Fig. 5 shows the detailed operation process for a single S-box, where the multiplication operations in $GF(2^4)$ and $GF(2^2)$ are both implemented using (4, 8) masking scheme. In detail, the process is initiated by mapping the masking shares in $GF(2^8)$ to the masking shares in $GF(2^4)$. Then, the masking shares of the inverse element in $GF(2^4)$ are calculated using the $GF(2^4)$Multiplier (the multiplication operation in $GF(2^4)$ is implemented using (4, 8) masking scheme) and the $GF(2^4)$Squaring circuit which performs the operation of squaring in $GF(2^4)$. In order to further reduce the complexity of calculation, the masking shares in $GF(2^4)$ are mapped to the masking shares in $GF(2^2)$. Similarly, the masking shares of the inverse element in $GF(2^2)$ are calculated using $GF(2^2)$Multiplier (again using (4, 8) masking scheme) and $GF(2^2)$Squaring for carrying out the squaring operation in $GF(2^2)$. Finally, the masking shares of the inverse element in $GF(2^2)$ are inversely mapped back to the masking shares in $GF(2^8)$ to accomplish the Subbytes operation. This last operation is performed in two sequential steps and corresponds to the lower half in Fig. 5.

Status register module: The main task of this module is to implement the storage of intermediate sharing masks, which are outputs of certain modules, and to perform the shift operation. The internal flow of data of the status register is shown in Fig. 6. Each S_{ij} contains three masking shares for each byte of the intermediate state, thus in total comprising 384 masked state bits. The multiplexer selects between the two inputs which are then processed by the status register. That is, it selects between *Input1* which contains three output masking shares of the SubBytes module and *Input2* that consists of three masking shares obtained by the MixColumns module for each column (thus $4 \times 3 \times 8 = 96$ bits in total). The status register is updated from the 5th to the 20th clock cycle, where in each clock three output masking shares that correspond to one of the 16 S-boxes are shifted through the state register as depicted in Fig. 6. During the 21st cycle, the ShiftRows operation is performed which is shown by blue arrows. When *Input2* is needed to be stored, the content of the memory blocks in each row is moved upwards and the data of the first row is moved to the MixColumns module. Finally, the last row containing S_{3j} is updated by moving the data of *Input2* there. This operation is performed from the 22th to the 25th clock cycle.

Key register module: The main objective of this module is to generate the key expansion of the masking shares. Similarly to the status register, each cell (block) in the key register stores three masking shares of the key. During the first 16th clock cycles, the masking shares of the round key are sequentially moved to the corresponding cells. Then, from the 17th to the 20th clock cycle, the masking shares in the last column $K_{03}, \ldots, K_{33}$ are processed. More specifically, three masking shares in K_{13} are used as the input to the SubBytes module whereas the remaining cells are simply shifted as follows: $K_{13} \rightarrow K_{03} \rightarrow K_{33} \rightarrow K_{23} \rightarrow K_{13}$. Consequently, the three masking shares contained in K_{13}, K_{23}, K_{33} and K_{03} are supplied to SubBytes module, in this specific order, during the consecutive four clock cycles. From the 21st to the 24th clock cycle, the key bytes (three masking shares) of the first column of the next round key are generated using the result of SubBytes module, the data of K_{03} and the round constants. At the same time, the key bytes K_{00}, K_{10}, K_{20} and K_{30} are moved so that $K_{00} \rightarrow K_{30} \rightarrow K_{20} \rightarrow K_{10} \rightarrow K_{00}$. For instance, the three masking shares of K_{13} are obtained in the 21st clock cycle from the SubBytes module. These masking shares are XORed with the data of the K_{00} and the round constant, and the result is stored in K_{30}. At the same time, the data of the K_{00}, K_{10}, K_{20} and K_{30} is shifted so that $K_{00} \rightarrow K_{30} \rightarrow K_{20} \rightarrow K_{10} \rightarrow K_{00}$. This process is then repeated during the next three cycles to generate masking shares for K_{23}, K_{33} and K_{03}. The key bytes for the remaining three columns are obtained

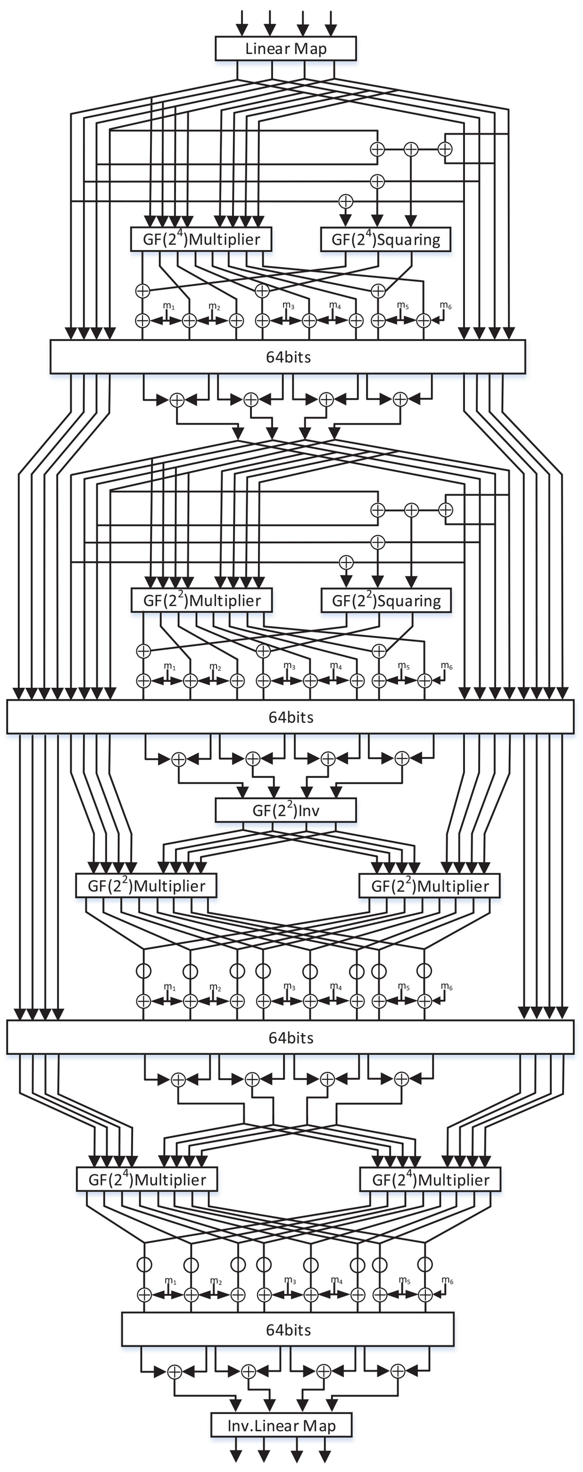

Fig. 5 *Structure diagram of the transition of the variables from the finite field $GF(2^8)$ to the finite field $GF(2^2)$*

from the 25th to 27th clock cycle. The process of key expansion is depicted in Fig. 7.

Fresh random number generator module: The main task of this module is to generate some fresh random values needed in the SubBytes module.

5 Performance evaluation of our implementation

In this section, we verify the security and evaluate the chip area of the new masking scheme on the Synopsys Design Compiler 2016.03 using the library NanGate 45 nm Open Cell Library [21].

5.1 SCA evaluation

To verify that our masking scheme can efficiently resist second-order DPA attacks, we have performed such an attack that uses correlation coefficients for AES in two different scenarios. That is, we first consider AES without any protection against DPA and then perform DPA on AES that employs our second-order masking scheme. During the synthesis, we use the KEEP_HIERARCHY option to prevent optimisations which would affect our design structure. The specific attack steps are given as follows.

i. *Power trace acquisition*: The power consumption of AES is collected using an oscilloscope and is performed for AES without any protection (Fig. 8) and AES that uses our second-order masking scheme generated by the SAKURA-G board (Fig. 9). The oscilloscope model is LeCroy WaveRunner 610Zi with sampling frequency of 250 MSamples/s. In Fig. 8, we can easily identify 11 distinct spikes, where the first 10 spikes refer to the execution of 10 round functions and the last spike corresponds to the process of generating ciphertext output. Fig. 9 shows the power consumption of AES with second-order masking scheme, having these spikes greatly covered by a noise.
ii. *Generate second-order power trace*: In our attack, we analyse XORing of two output masking shares of the S-boxes in the last round, where the collected power traces are used to generate the second-order power traces.
iii. *The resistances against second-order DPA*: We now perform second-order DPA attack to recover a single byte of the key by collecting power traces in the 10th round of AES using the so-called Hamming weight as a power leakage model. The correlation between 256 possible keys of the first key byte and the actual key byte is depicted in Figs. 10 and 11 for AES without and with masking, respectively. In Fig. 10, only one guessed key byte (the red trace) with the value 243(0xF3) has a higher correlation than other guessed keys, even if we increase the number of sampling traces. In our simulation, the actual key byte is exactly 243(0xF3), which is consistent with the guessed key byte. On the other hand, in Fig. 11, all the guessed keys bytes have almost the same correlation traces so that we cannot distinguish which guessed key byte is the correct key byte.

To recover all the subkeys in the last round of AES without any protection, we only need to collect 1000 power traces. This is depicted in Fig. 12, where the distinct 16 spikes identify the correct guessed values of 16 bytes secret key used in the last round. However, in the second-order masking scenario, we cannot distinguish the correct key bytes even if we collect 10 million power traces (see Fig. 13). This means that all guessed keys leave very similar correlations in terms of power traces and therefore it is impossible to distinguish the correct one. Hence, based on the above simulation results, we can conclude that our masking scheme provides high resistance against second-order DPA attacks.

5.2 Performance evaluation

In this section, the performance of our second-order masking scheme is evaluated with respect to the amount of random numbers, used chip area and the number of clock cycles (see Table 1). In Table 2, we compare our results to previous implementations of the second-order masking schemes in terms of the above parameters.

Randomness: The increase of the number of shares from 3 to 4 at the input of the SubBytes module requires an additional 8-bit random value. In the SubBytes module, a total number of $108 = 6 \times 4 + 6 \times 2 + 6 \times 4 + 6 \times 8 = 108$ random bits need to be added (cf. Fig. 5) to ensure the uniformity of the output masking shares after the multiplication operation. Therefore, a total number of random bits per round for our masking scheme is 116, which is 10 bits less than the work of [18] and significantly less (about

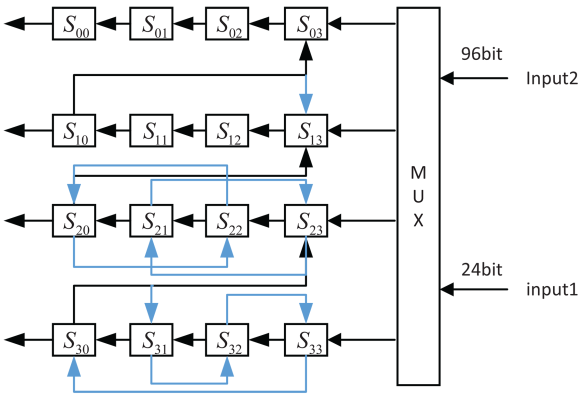

Fig. 6 *Internal structure diagram of status registers*

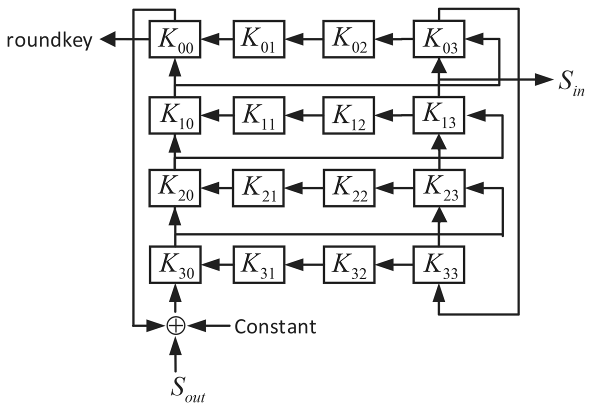

Fig. 7 *Structure diagram of the key registers*

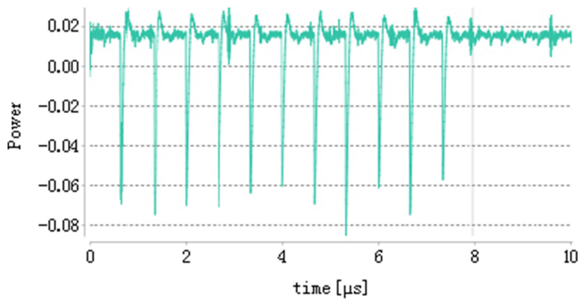

Fig. 8 *Power traces for AES without any protection*

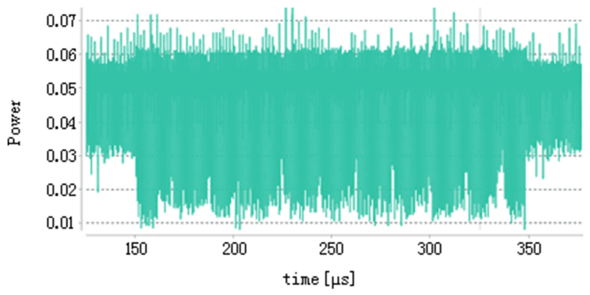

Fig. 9 *Power traces for AES using our second-order masking scheme*

28.4%) compared to implementation in [19] which uses 46 bits in extent but requires a slightly smaller chip area.

Area: The total chip area of our masking scheme is 11.9 kGE, which is slightly larger (∼13.7%) than the area in [19].

Clock cycles: In our scheme, five clock cycles are needed to store the data after each multiplication operation in the SubByte module. It is one clock cycle less compared to the works of [18, 19]. Note that the four input masking shares of the SubBytes module are independent and uniform, which implies that it is not necessary to register the data after the linear mapping. Moreover, exactly 27 clock cycles are required to perform a round function

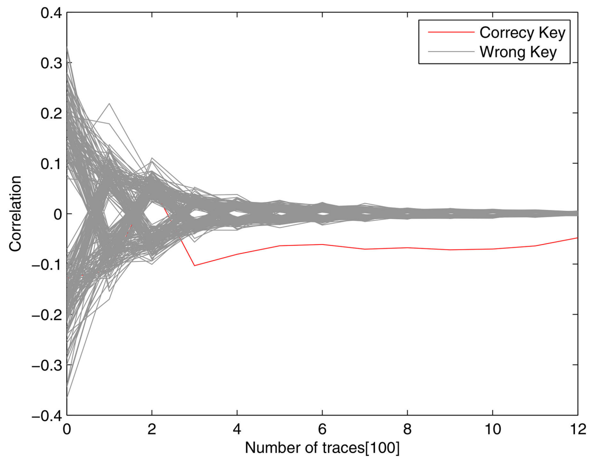

Fig. 10 *Identifying the first key byte without masking*

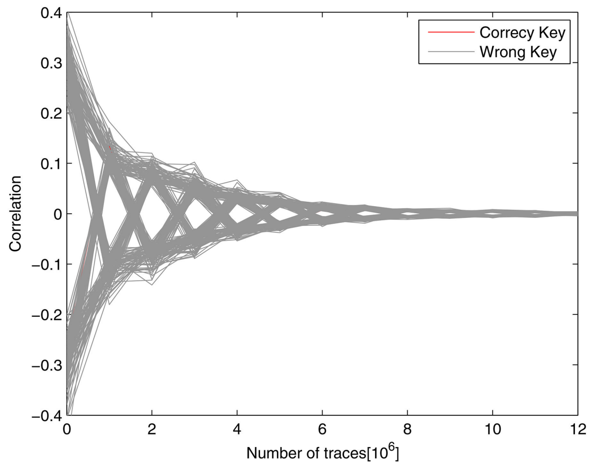

Fig. 11 *Identifying the first key byte with masking (no success)*

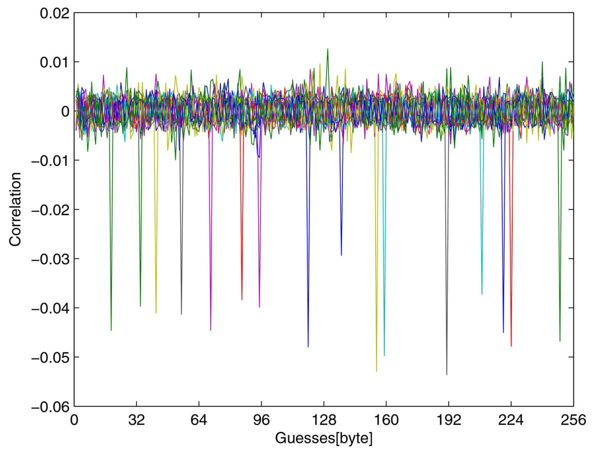

Fig. 12 *Recovering 16 bytes key without masking using 1000 traces*

operation, and therefore a total number of cycles to encrypt a 128-bit block of plaintext is $266 = 9 \times 27 + 23$ clock cycles, since the last round function of AES does not include the Mixcolumns operation which consumes four cycles in our implementation.

6 Conclusions

We have presented a second-order TI of AES, which employs some efficient first-order masking scheme. This approach provides a general design framework, thus allowing us to build efficient $(d+1)$th-order masking schemes based on the existing ones of order d. It is shown that only 116 random bits per round are required and the hardware implementation cost is about 11.9 kGE. Compared to the currently most efficient implementation due to De Cnudde *et al.* [19], our scheme reduces the use of fresh random bits

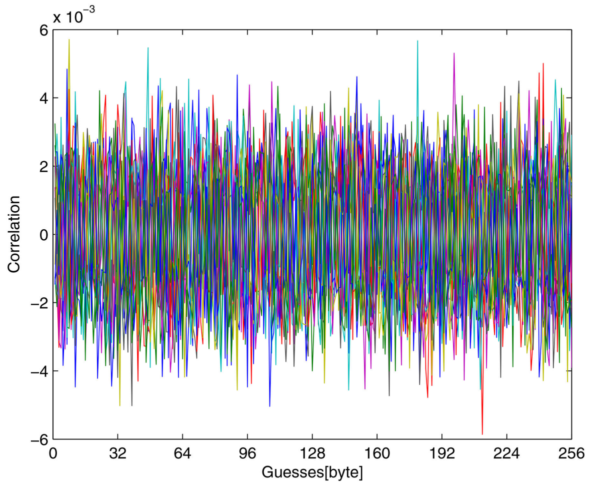

Fig. 13 *Power traces for 16 bytes key with masking*

Table 1 Area of different functions of the masked AES

	Area (GE)	
	Compile	Compile ultra
state register module	3102.1	3004.2
key register module	3556.7	3426.8
S-box module	4301.2	3860.21
MixColumn module	517.1	517.1
control module	138.85	127.74
MUX	252.7	252.7
total	11,876.4	11,195.5

Table 2 Performance comparison of different second-order masking schemes of AES

Resource	Randomness (bit)	Sbox area (GE)[a]	Area (GE)[a]	Clock cycle
[18]	126	11,174/7849	18,602/14,872	276
[19]	162	3796/3662	10,449/10,276	276
new	116	4301.2/3860.21	11,876.4/11,195.5	266

[a]Using compile/compile-ultra synthesis option.

by 28.4%, whereas the total chip area is only slightly increased (by 13.7%). Furthermore, the security of our masking scheme used to protect AES-128 against second-order DPA attacks has also been confirmed in a practical attack environment.

7 Acknowledgments

The works of Y. Wei and F. Yao were supported in part by the National Key R&D Program of China (no. 2017YFB0802000) and in part by the Natural Science Foundation of China (nos. 61572148, 61872103, 61572146, U1501252, U1711263). The work of A. Wang was supported in part by National Cryptography Development Fund (no. MMJJ20170201), in part by Beijing Natural Science Foundation (no. 4162053).

8 References

[1] Koeune, F., Standaert, F.: 'A tutorial on physical security and side-channel attacks'. Foundations of Security Analysis and Design III: FOSAD 2004/2005, New York, NY, USA, 2005 (LNCS, **3655**), pp. 78–108

[2] Kocher, P., Jaffe, J., Jun, B.: 'Differential power analysis'. Advances in Cryptology-CRYPTO 1999, Santa Barbara, California, USA, 1999 (LNCS, **1666**), pp. 388–397

[3] Daemen, J., Rijmen, V.: '*The design of Rijndael: AES-the advanced encryption standard*' (Springer Science & Business Media, USA, 2013)

[4] Akkar, M., Giraud, C.: 'An implementation of DES and AES, secure against some attacks'. Cryptographic Hardware and Embedded Systems-CHES 2001, Paris, France, 2001 (LNCS, **2162**), pp. 309–318

[5] Prouff, E., Rivain, M., Bevan, R.: 'Statistical analysis of second order differential power analysis', IACR Cryptology ePrint Archive, 2010, **646**, Available at https://eprint.iacr.org/2010/646.pdf

[6] Schramm, K., Paar, C.: 'Higher order masking of the AES'. Topics in Cryptology-CT-RSA 2006, San Jose, CA, USA, 2010 (LNCS, **3860**), pp. 208–225

[7] Chari, S., Jutla, C., Rao, J., *et al.*: 'Towards sound approaches to counteract power-analysis attacks'. Advances in Cryptology-CRYPTO 1999, Santa Barbara, California, USA, 1999 (LNCS, **1666**), pp. 398–412

[8] Goubin, L., Patarin, J.: 'DES and differential power analysis the 'duplication' method'. Cryptographic Hardware and Embedded Systems-CHES 1999, Worcester, MA, USA, 1999 (LNCS, **1717**), pp. 158–172

[9] Messerges, T.: 'Securing the AES finalists against power analysis attacks'. Fast Software Encryption-FSE 2000, New York, NY, USA, 2001 (LNCS, **1978**), pp. 150–164

[10] Mangard, S., Pramstaller, N., Oswald, E.: 'Successfully attacking masked AES hardware implementations'. Cryptographic Hardware and Embedded Systems-CHES 2005, Edinburgh, UK, 2005 (LNCS, **3659**), pp. 157–171

[11] Mangard, S., Popp, T., Gammel, B.: 'Side-channel leakage of masked CMOS gates'. Topics in Cryptology-CT-RSA 2005, San Francisco, CA, USA, 2005 (LNCS, **3376**), pp. 351–365

[12] Moradi, A., Mischke, O., Eisenbarth, T.: 'Correlation-enhanced power analysis collision attack'. Cryptographic Hardware and Embedded Systems-CHES 2010, Santa Barbara, California, USA, 2010 (LNCS, **6225**), pp. 125–139

[13] Nikova, S., Rechberger, C., Rijmen, V.: 'Threshold implementations against side-channel attacks and glitches'. Information and Communications Security-ICICS 2006, Raleigh, NC, USA, 2006 (LNCS, **4307**), pp. 529–545

[14] Fumaroli, G., Martinelli, A., Prouff, E.: 'Affine masking against higher-order side channel analysis'. Selected Areas in Cryptography-SAC 2010, Waterloo, Ontario, Canada, 2010 (LNCS, **6544**), pp. 262–280

[15] Ishai, Y., Sahai, A., Wagner, D.: 'Private circuits: securing hardware against probing attacks'. Advances in Cryptology-CRYPTO 2003, Santa Barbara, California, USA, 2003 (LNCS, **2729**), pp. 463–481

[16] Rivain, M., Prouff, E.: 'Provably secure higher-order masking of AES'. Cryptographic Hardware and Embedded Systems-CHES 2010, Santa Barbara, California, USA, 2010 (LNCS, **6225**), pp. 413–427

[17] Bilgin, B., Gierlichs, B., Nikova, S.: 'A more efficient AES threshold implementation'. Progress in Cryptology-AFRICACRYPT 2014, Africa, Marrakesh, Morocco, 2014 (LNCS, **8469**), pp. 267–284

[18] De Cnudde, T., Bilgin, B., Reparaz, O.: 'Higher-order threshold implementation of the AES S-box'. Smart Card Research and Advanced Applications-CARDIS 2015, Paris, France, 2015 (LNCS, **9514**), pp. 259–272

[19] De Cnudde, T., Reparaz, O., Bilgin, B.: 'Masking AES with $d+1$ shares in hardware'. Cryptographic Hardware and Embedded Systems-CHES 2016, Santa Barbara, California, USA, 2016 (LNCS, **9813**), pp. 194–212

[20] Gross, H., Mangard, S., Korak, T.: 'Domain-oriented masking: compact masked hardware implementations with arbitrary protection order', IACR Cryptology ePrint Archive, 2016, **486**, Available at https://eprint.iacr.org/2016/486.pdf

[21] 'NanGate Open Cell Librar'. Available at http://www.nangate.com/

[22] Duc, A., Dziembowski, S., Faust, S.: 'Unifying leakage models: from probing attacks to noisy leakage'. Advances in Cryptology-EUROCRYPT 2014, Copenhagen, Denmark, 2014 (LNCS, **8441**), pp. 423–440

[23] Bilgin, B.: '*Threshold implementations: as countermeasure against higher-order differential power analysis*' (University of Twente, Netherlands, 2015)

[24] Canright, D.: 'A very compact S-box for AES'. Cryptographic Hardware and Embedded Systems-CHES 2005, Edinburgh, UK, 2005 (LNCS, **3659**), pp. 441–455