Inner Product Masking for Bitslice Ciphers and Security Order Amplification for Linear Leakages

Weijia Wang[1], François-Xavier Standaert[2], Yu Yu[1,3],
Sihang Pu[1], Junrong Liu[1], Zheng Guo[1], and Dawu Gu[1]

[1] School of Electronic Information and Electrical Engineering,
Shanghai Jiao Tong University, China
Email: {aawwjaa,yyuu,push.beni,liujr,guozheng,dwgu}@sjtu.edu.cn
[2] ICTEAM/ELEN/Crypto Group, Université catholique de Louvain, Belgium
Email: fstandae@uclouvain.be
[3] Westone Cryptologic Research Center

Abstract. Designers of masking schemes are usually torn between the contradicting goals of maximizing the security gains while minimizing the performance overheads. Boolean masking is one extreme example of this tradeoff: its algebraic structure is as simple as can be (and so are its implementations), but it typically suffers more from implementation weaknesses. For example knowing one bit of each share is enough to know one bit of secret in this case. Inner product masking lies at the other side of this tradeoff: its algebraic structure is more involved, making it more expensive to implement (especially at higher orders), but it ensures stronger security guarantees. For example, knowing one bit of each share is not enough to know one bit of secret in this case.

In this paper, we try to combine the best of these two worlds, and propose a new masking scheme mixing a single Boolean matrix product (to improve the algebraic complexity of the scheme) with standard additive Boolean masking (to allow efficient higher-order implementations). We show that such a masking is well suited for application to bitslice ciphers. We also conduct a comprehensive security analysis of the proposed scheme. For this purpose, we give a security proof in the probing model, and carry out an information leakage evaluation of an idealized implementation. For certain leakage functions, the latter exhibits surprising observations, namely information leakages in higher statistical moments than guaranteed by the proof in the probing model, which we can connect to the recent literature on low entropy masking schemes. We conclude the paper with a performance evaluation, which confirms that both for security and performance reasons, our new masking scheme (which can be viewed as a variation of inner product masking) compares favorably to state-of-the-art masking schemes for bitslice ciphers.

1 Introduction

In the recent literature on masking schemes, increasing the algebraic complexity of the operation mixing the shares has been frequently used to improve the

resistance of cryptographic implementations in scenarios where limited noise is available in the adversary's measurements. Examples include inner product masking [1], polynomial masking [7, 14] and affine masking [6]. For a comparable amount of shares, these masking schemes offer a (sometimes slightly) better security than the mainstream Boolean masking. Yet, this usually comes at the cost of (sometimes large) performance overheads.

In this paper, we therefore start from the observation that it would be interesting to design a hybrid masking scheme, where some of the shares are mixed based on a more complex operation (to guarantee some security in low noise contexts where the simplicity of Boolean masking is problematic), while the others are just mixed thanks to additive Boolean masking (which efficiently generalizes to higher-orders [15]). We instantiate a first proposal in this direction, that we denote as Boolean matrix product masking, and which is particularly well suited to block ciphers with efficient bitslice representation. In this masking scheme, we split the secret x into n shares $\boldsymbol{x} = (x_1, \ldots, x_n)$ and use a (public) random nonsingular Boolean matrix A, such that $x = A \times x_1 \oplus \bigoplus_{2 \leq i \leq n} x_i$ (see Section 2.1 for definitions and notations) and the public matrix A is fixed as a constant in each running of the masked block cipher. Intuitively, our masking scheme can therefore be seen as a variant of inner product masking specialized to bitslice ciphers (yet applicable to the AES), which we discuss in Sections 2.6 and 2.7. Note that inner product masking is itself a particular case of the code-based masking recently introduced in [4, 3]. Next, we show how to perform standard operations such as addition and multiplication (i.e. bitwise XOR and AND in GF(2)) securely. The standard operations can be composed to protect a complete bitslice cipher such as the recently introduced (X)LS-designs [8, 11].

We then investigate the security of our masking scheme in the probing model [10], and prove that it guarantees dth-order security for $n \geq 2d + 1$. Further, we evaluate the concrete information leakage of our masking shares based on an information theoretic analysis [16]. As expected, the results show that they leak less than Boolean shares and comparably to inner product shares in low noise contexts. More surprisingly, the information theoretic analysis also reveals that in high noise contexts, Boolean matrix product masking (and, in fact, inner product masking in general) can lead to additional gains. Namely, their information leakages can be of higher order than what is guaranteed by the proof in the probing model. As in the context of low entropy masking scheme, we can justify that this gain can only be observed for linear leakage functions [9].

We finally complement these results with a performance evaluation, which allows us to complete the picture of our new masking scheme. In particular, the implementations of the masked LS-design Fantomas exhibit excellent performances, with only limited overheads compared to Boolean masking.

2 Our Construction

In this section, we give the construction of our masking scheme, including the encoding and decoding of the secret and different operations in masked domain.

2.1 Preliminaries

Let lowercases (e.g., i, x) denote any integral variables or binary vectors, and capital letters (e.g., A) be the Boolean matrices. $A(i,:)$ (or $A(i)$ for short) denotes the ith row of matrix A and $x(i)$ denotes the ith element of vector x. And $A(:,i)$ denotes the ith column of matrix A. Let $A(i:j,k)$ (resp., $A(k,i:j)$) be the elements of kth column (resp., kth row) and ith to jth row (resp., ith to jth column). Let the bold lowercases (e.g., $\boldsymbol{x} = (x_1, x_2, \ldots)$) denote the vectors whose elements are binary vectors, and let the bold capital letters be vectors of Boolean matrices (e.g., $\boldsymbol{X} = (X_1, X_2, \ldots)$)). Finally, Let E denote the identity matrix and A^{-1} and A^{T} denote the inverse and transpose of the matrix A, respectively. We recall the tensor product (denoted as $\otimes$) between Boolean matrices. Suppose that two matrices are A, B with size $m_1 \times n_1$ and $m_2 \times n_2$ respectively, then the result is a matrix C with size $m_1 m_2 \times n_1 n_2$:

$$A \otimes B \stackrel{\mathsf{def}}{=} \begin{bmatrix} A(1,1) & \ldots, & A(1,n_1) \\ \ldots & \ldots, & \ldots \\ A(m_1,1) & \ldots, & A(m_1,n_1) \end{bmatrix} \otimes \begin{bmatrix} B(1,1) & \ldots, & B(1,n_2) \\ \ldots & \ldots, & \ldots \\ B(m_2,1) & \ldots, & B(m_2,n_2) \end{bmatrix}$$

$$= \begin{bmatrix} A(1,1)B & \ldots, & A(1,n_1)B \\ \ldots & \ldots, & \ldots \\ A(m_1,1)B & \ldots, & A(m_1,n_1)B \end{bmatrix} = C,$$

where $A(i,j)B = \begin{bmatrix} A(i,j)B(1,1) & \ldots, & A(i,j)B(1,n_2) \\ \ldots & \ldots, & \ldots \\ A(i,j)B(m_2,1) & \ldots, & A(i,j)B(m_2,n_2) \end{bmatrix}$.

2.2 Encoding & Decoding of the secret variable

The encoding of an m-bit secret variable (say, x) is close to that of the Boolean masking but the first share is multiplied by a nonsingular matrix: $x = (A \times x_1) \oplus x_2 \oplus \ldots \oplus x_n$. Algorithms 1 and 2 are the pseudocode of encoding and decoding respectively. Note that the matrix A (and its inverse) is fixed in each running of the block cipher, thus in the remainder of this paper, we often omit the matrix A and use the shares $\boldsymbol{x} = (x_1, \ldots, x_n)$ to represent the encoding of x.

Algorithm 1 Enc

Require: m-bit secret variable x, invertible matrix A and its inverse A^{-1}
Ensure: $\mathrm{Enc}(x) = \boldsymbol{x} = (x_1, \ldots, x_n)$ as the masked variables
1: **for** $i = 1$; $i < n$; $i{+}{+}$ **do**
2: $\quad x_{i+1}$ is a randomly generated m-bit value
3: **end for**
4: $x_1 := A^{-1} \times (x \oplus \bigoplus_{i=2}^{n} x_i)$

Algorithm 2 Dec

Require: masked variables $\boldsymbol{x} = x_1, \ldots, x_n$
Ensure: $x = \mathrm{Dec}(\boldsymbol{x})$ as the secret variable
1: $x := (A \times x_1) \oplus \bigoplus_{i=2}^{n} x_i$

2.3 Initialization for masked operations

In order to reduce the complexity of masked operations, an initialization step is necessary to pre-compute some variables. Algorithm 3 details the pre-computation of the matrices (A, A^{-1}), $\hat{A}$, $\grave{A}$ and $\acute{A}$, where $\otimes$ is tensor product, ReRandMat() is the function that re-randomize the nonsingular matrix and its inverse (details deferred to Section 2.5), A_{old} and A_{old}^{-1} are the Boolean matrices of the last running (which can been replaced by the identity matrix for the first run), and E' consists of the $(i + m(i-1))$-th rows of the $m \times m^2$ identity matrix for all $i \in \{1, \ldots, m\}$. For example, we have: $E' = \begin{bmatrix} 1,0,0,0,0,0,0,0,0 \\ 0,0,0,0,1,0,0,0,0 \\ 0,0,0,0,0,0,0,0,1 \end{bmatrix}$ for $m = 3$.

Algorithm 3 Setup

Require: length of the masked variable m
Ensure: random nonsingular $m \times m$ matrix A (and its inverse A^{-1}) and some other pre-computed values
1: $(A, A^{-1}) := \mathrm{ReRandMat}(\mathrm{A_{old}}, \mathrm{A_{old}^{-1}})$
2: $\hat{A} := A^{-1} \times (E' \times (A \otimes A))$
3: $\grave{A} := A^{-1} \times (E' \times (A \otimes E))$
4: $\acute{A} := A^{-1} \times (E' \times (E \otimes A))$

2.4 Operations in masked domain

Mask refreshing. Mask refreshing is a re-randomized procedure to re-encode the secret variables. As we will introduce in Section 3.1, this procedure will be called n times at the beginning of block cipher to re-randomize the masking of the key. Algorithm 4 gives the details of this operation.

Addition (XOR) of two masked variables. Algorithm 5 gives the masked addition of $\boldsymbol{x} = (x_1, \ldots, x_n)$ and $\boldsymbol{y} = (y_1, \ldots, y_n)$ that are encodings of two secret values x and y respectively.

Bitand of two masked variables. The most basic nonlinear operation for bitslice S-box is the bitand. This operation in masked domain is given in Algorithm 6, where variables, x and y, are encoded as $\boldsymbol{x} = (x_1, \ldots, x_n)$ and

Algorithm 4 Refresh

Require: masked variable $\boldsymbol{x} = (x_1, \ldots, x_n)$
Ensure: refreshed encoding $\boldsymbol{x'} = (x'_1, \ldots, x'_n)$
1: randomly generate a vector of m-bit variables $\boldsymbol{a} = (a_1, \ldots, a_n)$ s.t. $(A \times a_1) \oplus a_2 \oplus \ldots \oplus a_n = 0$
2: **for** $i = 1$; $i \leq n$; $i{+}{+}$ **do**
3: $\quad x'_i := x_i \oplus a_i$
4: **end for**

Algorithm 5 SecAdd

Require: two masked variables $\boldsymbol{x} = (x_1, ..., x_n)$ and $\boldsymbol{y} = (y_1, ..., y_n)$
Ensure: encoding of $x \oplus y$ (namely, $(\text{Enc}(x \oplus y) = \boldsymbol{z} = (z_1, ..., z_n))$
1: **for** $i = 1$; $i \leq n$; $i{+}{+}$ **do**
2: $\quad z_i := x_i \oplus y_i$
3: **end for**

$\boldsymbol{y} = (y_1, \ldots, y_n)$ respectively, and $\odot$ denotes bitand. The algorithm is similar in spirit to the ISW scheme [10] except for some additional adaptions to the matrix case. Note that $x_i \otimes y_i$ is of size $m^2 \times 1$ (by tensor product $\otimes$), and the (pre-computed) matrices $\hat{A}$, $\grave{A}$ and $\acute{A}$ are all of size $m \times m^2$ (see Algorithm 3). Thus $t_{i,j}$ is always of size $m \times 1$. We next sketch the proof of correctness for Algorithm 6. First we have:

$$
\begin{aligned}
&(A \times x_1 \oplus x_2 \oplus \ldots \oplus x_n) \odot (A \times y_1 \oplus y_2 \oplus \ldots \oplus y_n) \\
&= ((A \times x_1) \odot (A \times y_1)) \oplus ((A \times x_1) \odot y_2) \oplus \ldots \oplus ((A \times x_1) \odot y_n) \\
&\oplus (x_2 \odot (A \times y_1)) \qquad \oplus (x_2 \odot y_2) \qquad \oplus \ldots \oplus (x_2 \odot y_n) \\
&\ldots \\
&\oplus (x_n \odot (A \times y_1)) \qquad \oplus (x_n \odot y_2) \qquad \oplus \ldots \oplus (x_n \odot y_n) \ .
\end{aligned}
$$

We handle the terms separately, let $u = (A \times x_1) \odot (A \times y_1)$, then we have: $u(i) = \bigoplus_{i,j \in (1,\ldots,m)} A(i,j) x_1(i) y_1(j) = (A(i,:) \otimes A(i,:)) \times (x_1 \otimes y_1)$. Thus we have $t_{i,j} = A^{-1} \times (A \times x_1) \odot (A \times y_1) = \hat{A} \times (x_1 \otimes y_1)$. Similar conclusions can be obtained for $A^{-1} \times (A \times x_1) \odot y_i$ and $A^{-1} \times x_i \odot (A \times x_1)$ when $i \in (2, \ldots, n)$. Therefore we can prove that $(A \times x_1 \oplus x_2 \oplus \ldots \oplus x_n) \odot (A \times y_1 \oplus y_2 \oplus \ldots \oplus y_n) = (\bigoplus_{i,j \in (2,\ldots,m)} t_{i,j}) \oplus (\bigoplus_{j \in (1,\ldots,m)} A \times t_{1,j}) \oplus (\bigoplus_{i \in (1,\ldots,m)} A \times t_{i,1})$. Finally the lines 6-19 are very similar to the ISW scheme and please refer to [10] for the remainder of correctness proof.

Note that Algorithm 6 includes the multiplication operation between a fix matrix (size of $m \times m^2$ or $m \times m$) and a scalar, which is not very efficient with the processors that don't support the 'popcnt' instruction.[4] Thus we present in Algorithms 7, 8 and 9 different ways of computing the multiplication between a matrix A (size of $m_1 \times m_2$) and a scalar x for different situations. Algorithm 7 benefits from the 'popcnt' instruction and its time / memory complexities

[4] 'popcnt' instruction counts the number of bits set to 1 in one cycle

are $O(m_1 * m_2/w)$ / $O(1)$, where w is the bit width of the processor and we only consider the case that $w|m_1$ and $w|m_2$. Algorithm 8 first operates the bitand between each line of A and x, resulting in a matrix V of size $m_1 \times m_2$, then it computes the product y in a bitslice manner by XORing the columns of V. Its time complexity is also $O(m_1 * m_2/w)$ but the memory complexity is $O(m_1 * m_2)$ for the storage of matrix V. Thus Algorithm 8 can obtain a same time complexity without the supporting of 'popcnt' instruction at the cost of some (but not much) memory complexity. As show in Figure 1, Algorithm 9 first separates A and x into k equal sized parts (each of length is $l = m_2/k$) and creates the look-up table $M_i()$ for each part of scalar (i.e., $x((i-1)*l+1 : i*l)$) multiplied by the corresponding part of matrix (i.e., $A(:, (i-1)*l+1 : i*l)$). As presented before, the matrix A should be fixed during each running of the encryption / decryption, thus these look-up tables can be pre-computed in the setup stage and stored in the memory or flash. Finally the multiplication can be done by XORing the result of k times table look-up. The time complexity of Algorithm 9 is $O(k * m_1/w)$ excluding the pre-computing of look-up tables, but the memory complexity is relatively larger: $O(k * m_1 * 2^{m_2/k})$. In particular, if we take $k = m_2$, we then have the same time complexity as Algorithm 8 but double the memory. Note that, thanks to the table look-up process, there are less variables operated in Algorithm 9, thus it can be more secure than the other two against multivariate side-channel attacks.

Algorithm 6 SecBitAnd

Require: $\boldsymbol{x} = (x_1, \ldots, x_n)$, $\boldsymbol{y} = (y_1, \ldots, y_n)$,$\hat{A}$, $\grave{A}$, $\acute{A}$
Ensure: encoding of $x \odot y$ (i.e., $\boldsymbol{z} = (z_1, \ldots, z_n)$)

1: **for** $i = 1$; $i \leq n$; i++ **do**
2: **for** $j = 1$; $j \leq n$; j++ **do**
3: $$t_{i,j} := \begin{cases} \hat{A} \times (x_i \otimes y_j) & \text{if } i = j = 1 \\ \grave{A} \times (x_i \otimes y_j) & \text{if } i = 1 \\ \acute{A} \times (x_i \otimes y_j) & \text{if } j = 1 \\ x_i \odot y_j & \text{others} \end{cases}$$
4: **end for**
5: **end for**
6: Let a matrix of vectors $\boldsymbol{T} = (t_{i,j})$
7: **for** $i = 1$; $i \leq n$; i++ **do**
8: $r_{i,i} := t_{i,i}$
9: **for** $j = i + 1$; $j \leq n$; i++ **do**
10: Set $r_{i,j}$ to be a random $m \times 1$ vector
11: $r_{j,i} := t_{j,i} + (r_{i,j} + t_{i,j})$
12: **if** $i = 1$ **then**
13: $r_{i,j} := A \times r_{i,j}$
14: **end if**
15: **end for**
16: **end for**
17: **for** $i = 1$; $i \leq n$; i++ **do**
18: $z_i := \bigoplus_j r_{j,i}$
19: **end for**

Algorithm 7 MatrixMul-popcnt

Require: an $m_1 \times m_2$ matrix A, an $m_2 \times 1$ scalar x
Ensure: $y = A \times x$
1: **for** $i = 1$; $i \leq m_1$; i++ **do**
2: $v = A(i,:) \odot x^{\mathrm{T}}$
3: $y(i) = \mathrm{popcnt(v)}$
4: **end for**

Algorithm 8 MatrixMul-bitslice

Require: an $m_1 \times m_2$ matrix A
Ensure: $y = A \times x$
1: **for** $i = 1$; $i \leq m_1$; i++ **do**
2: $V(i,:) = A(i,:) \odot x^{\mathrm{T}}$
3: **end for**
4: $y = V(:,1)$
5: **for** $j = 2$; $j \leq m_2$; j++ **do**
6: $y = y \oplus V(:,j)$
7: **end for**

Algorithm 9 MatrixMul-tabulate

Require: an $m_1 \times m_2$ matrix A, k that we have $k|m_2$
Ensure: pre-computed tables $\mathrm{M}_{i \in \{1,\ldots,k\}}$
1: **Pre-computation Stage:**
2: $l = m_2/k$
3: **for** $i = 1$; $i \leq k$; i++ **do**
4: create the loop-up table M_i for any l-bit value multiplied by the matrix $A(:,(i-1)*l+1 : i*l)$, i.e., $\mathrm{M}_i(v) = A(:,(i-1)*l+1 : i*l) \times v$
5: **end for**

Require: an $m_2 \times 1$ scalar x, k and the pre-computed tables $\mathrm{M}_{i \in \{1,\ldots,k\}}$
Ensure: $y = A \times x$
1: **Online Stage:**
2: $l = m_2/k$
3: $y = M_1(x(1:l))$
4: **for** $i = 2$; $i \leq k$; i++ **do**
5: $y = y \oplus \mathrm{M}_i(x((l-1)*i+1 : l*i))$
6: **end for**

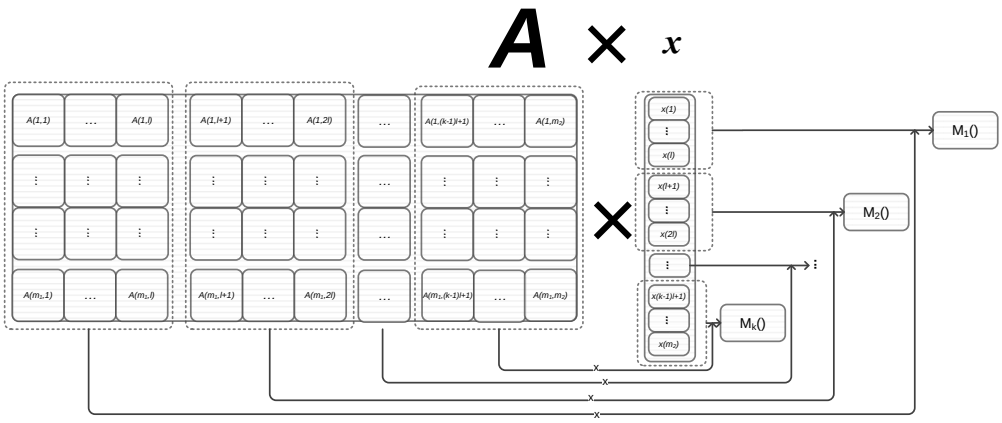

Fig. 1. Create the look-up table.

2.5 Nonsingular matrix re-randomization

We introduce in Algorithm 10 how to re-randomize a nonsingular matrix and its inverse in an efficient manner. A random elementary matrix T is generated using only rows switching from the identity matrix (lines 1, 2). It is obvious that this matrix is orthogonal, i.e., $T^{-1} = T^{\mathrm{T}}$. In each iteration of the loop (lines 5-7), a random elementary matrix P is generated using only one row addition with random bits and the corresponding row of A is re-randomized by multiplying with P, i.e., $P \times A$. We also keep a record of its inverse $A^{-1} \times P^{-1} = A^{-1} \times P$ (since P's inverse is P itself) along the way so that Algorithm 10 needs no matrix inverse operations. We admit that the output of Algorithm 10 is not strictly uniform over the set of all Boolean nonsingular matrices. But this is not a problem in our setting where we anyway assume that the matrix is fixed and public in each run of the masked block cipher.

Algorithm 10 ReRandMat

Require: A_{old} and A_{old}^{-1}
Ensure: $m \times m$ random matrices A, A^{-1}
1: $T := E$, $A := A_{old}$, $A^{-1} := A_{old}^{-1}$
2: Randomly permute the rows of T
3: $A := T \times A$, $A^{-1} := A^{-1} \times T^{\mathrm{T}}$
4: **for** $i = 1$; $i \leq m$; $i{+}{+}$ **do**
5: $P := E$
6: generate $m - 1$ random bits, replace the zeros in $P(i)$ with the random bits
7: $A := P \times A$, $A^{-1} := A^{-1} \times P$
8: **end for**

Based on the operations above, we can construct the masking scheme of bitslice block cipher. The description of masked (X)LS-design block ciphers is given in the full version due to lack of space.

2.6 Links with inner product masking

Our masking scheme can be seen as a variant of inner product masking, specialized to bitslice ciphers for efficiency purposes. Recall that the inner product masking shares a secret variable as $x = l_1 \cdot x_1 + l_2 \cdot x_2 + \ldots + l_n \cdot x_n$, where $\cdot$ denotes the multiplication in a Galois field, the vector $(l_1, \ldots, l_n)$ is public, and every $(n-1)$-tuple of $\{x_1, \ldots, x_n\}$ is independent of x. As the multiplication (in a Galois field) of two variables $x \cdot y$ can be represent as y left-multiplied by the x's GF(2^m)-multiplication matrix X, i.e., $x \cdot y = X \times y$, where the first column of X is x and the other ones are generated by $X(, i) = 2^i \cdot x$, the inner product masking can been rewritten as $x = A_1 \times x_1 + \ldots + A_n \times x_n$, where $A_1, \ldots, A_n$ are GF(2^m)-muliplication matrices. Thus the inner product masking can been seen as a masking such that (1) each share is multiplied by a different Boolean matrix and, (2) the matrices are selected as corresponding to Galois field multiplication rather than simply as nonsingular Boolean.

2.7 Application to the AES

The application of our masking to the AES is also possible. As the Galois field multiplication of two variables $x \cdot y$ can be represented as $x \cdot y = X \times y$, we define a function $\mathrm{F}_{\mathrm{GF}} : \mathbb{F}_2^m \to \mathbb{F}_2^m \times \mathbb{F}_2^m$, which converts any value in $\mathrm{GF}(2^m)$ to the corresponding Boolean matrix, i.e., $\mathrm{F}_{\mathrm{GF}}(x) = [2^0 \cdot x : 2^1 \cdot x : \ldots : 2^{m-1} \cdot x]$, where : concatenates the vectors (or matrices) of two sides thereof. Then the multiplication of two variables $\hat{x} = A_x \times x$ and $\hat{y} = A_y \times y$ equals:

$$\begin{aligned} z &= A_z^{-1} \times (\hat{x} \cdot \hat{y}) \\ &= A_z^{-1} \times (\hat{x} \cdot (A_y \times y)) \\ &= A_z^{-1} \times \mathrm{F}_{\mathrm{GF}(2^m)}(\hat{x}) \times A_y \times y \\ &= A_z^{-1} \times [F_{\mathrm{GF}}(2^0) \times \hat{x} : \ldots : F_{\mathrm{GF}}(2^{m-1}) \times \hat{x}] \times A_y \times y \\ &= [A_z^{-1} \times F_{\mathrm{GF}}(2^0) \times A_x \times x : \ldots : A_z^{-1} \times F_{\mathrm{GF}}(2^{m-1}) \times A_x \times x] \times A_y \times y \ . \end{aligned}$$

This process is precised in Algorithm 11. $G(, i)$ denotes the t-th column of binary matrix G, the list of matrices $(J_1, \ldots, J_m)$ can be hard-coded in the implementation and the corresponding $(H_1, \ldots, H_m)$ can be pre-computed in the setup phase. Therefore, the masked multiplication in Galois field can be constructed by modifying the line 3 of Algorithm 6 using Algorithm 11.

Algorithm 11 GFMul

Require: two variables x, y and the corresponding matrices A_x, A_y and A_z
Ensure: $z = A_z^{-1} \times ((A_x \times x) \cdot (A_y \times y))$
1: $(J_1, \ldots, J_m) := (F_{\mathrm{GF}}(2^0), \ldots, F_{\mathrm{GF}}(2^{m-1}))$
2: $(H_1, \ldots, H_m) := (A_z^{-1} \times J_1 \times A_x, \ldots, A_z^{-1} \times J_m \times A_x)$
3: **for** $i = 1$; $i <= m$; $i{+}{+}$ **do**
4: $\quad G(, i) := H_i \times x$
5: **end for**
6: $temp := G \times A_y$
7: $z := temp \times y$

3 Security Analysis

3.1 Provable security in the probing model

In this section, we give a security proof for our masking scheme in the probing model introduced in [10]. Recall that an m-bit variable x is encoded into n shares $(x_1, \ldots, x_n)$ with $x = (A \times x_1) \oplus x_2 \oplus \ldots \oplus x_n$ for a nonsingular public matrix A. We omit the leakage about A since it is public. We will show that our masking scheme is secure against d-probing adversary for $n \geq 2d + 1$. That is, every d-tuple of its intermediate variable is independent of any sensitive variables. In order to do this, we shall show that every d-tuple of its intermediate variable can be perfectly simulated without knowledge of any inputs.

Security for the masking operations We start the security analysis of the masked operations with a security proof for Algorithm 6.

Theorem 1 *Let $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ be two encodings from the inputs of Algorithm 6 and let $n \geq 2d + 1$. Then the distribution of every tuple of d intermediate variables in Algorithm 6 is independent of the distributions $x = A \times x_1 \oplus \bigoplus_{2 \leq i \leq n} x_i$ and $y = A \times y_1 \oplus \bigoplus_{2 \leq i \leq n} y_i$.*

Our proof follows and is very similar to the one outlined in [10]. We show that we can efficiently construct a $(n-1)$-tuple of random variables which is identically distributed to any d-tuple $(v_1, v_2, \ldots, v_d)$ of intermediate variables of Algorithm 6, independently of any statement about x and y. Therefore, we shall construct a set I of indices in $\{1, \ldots, n\}$ with cardinalities lower than or equal to $n-1$ and such that the distribution of any d-tuple $(v_1, v_2, \ldots, v_d)$ can be perfectly simulated from $x_{|I} \stackrel{\mathsf{def}}{=} (x_i)_{i \in I}$ and $y_{|I} \stackrel{\mathsf{def}}{=} (y_i)_{i \in I}$. This will prove the Theorem 1 since, $x_1, \ldots, x_n$ (and $y_1, \ldots, y_n$) being $(n-1)$-wise independent, $x_{|I}$ and $y_{|I}$ are jointly independent of $(x; y)$ as long as the cardinalities of I is strictly smaller than d, where I is constructed as follows:

1. Initially, I is empty and all v_h's are unassigned.
2. For every intermediate variables of the form x_i, y_i, $t_{i,i}$, $r_{i,j}$ (for any $i \neq j$), or a sum of values of above form (including z_i as a special case), add i to I. This covers all the intermediate variables except for ones corresponding to $t_{i,j}$ or $r_{i,j} + t_{i,j}$ for some $i \neq j$. For such variables, add both i and j to I.
3. Now that the set I has been determined, and cardinality of I can be at most $m = 2t$ since there are at most t intermediate variables. We show how to complete a perfect simulation of the values on intermediate variables using only values $x|_I$ and $y|_I$. Assign values to the $r_{i,j}$ as follows:
   - If $i \in I$ (regardless of j), then $r_{i,j}$ does not enter into the computation for any intermediate variables. Thus, its value can be left unassigned.
   - If $i \in I$, but $j \notin I$, then $r_{i,j}$ is assigned a random independent value. Analysis: Note that if $i < j$ this is what would have happened in the Algorithm 6. If $i > j$, however, we are making use of the fact that by construction, $r_{i,j}$ will never be used in the computation of any intermediate variables. Hence, we can treat $r_{i,j}$ as a uniformly random and independent value.
   - If both $i \in I$ and $j \in I$, then we have access to x_i, x_j, y_i and y_j. Thus, we compute $r_{i,j}$ and $r_{j,i}$ exactly as they would have been computed in the actual Algorithm 6.
4. For every intermediate variable of the form x_i, y_i, $x_i y_i$ (for any $i \neq j$), or a sum of values of the above form (including z_i as a special case), we know that $i \in I$, and all the needed values of $r_{i,j}$ have already been assigned in a perfect simulation. Thus, the intermediate variable can be computed in a perfect simulation.
5. The only types of intermediate variables left are $t_{i,j}$ or $r_{i,j} + t_{i,j}$. But by step 2, both $i, j \in I$, and by Step 3, $z_{i,j}$ has been assigned, thus the value of intermediate variable can be simulated perfectly.

6. Note that all z_i values for $i \in I$ can be simulated perfectly by the argument above. This completes the simulation and the argument of correctness.

Theorem 1 considers the probing of any variable in Algorithm 6 and Corollary 1 below states a result for the probing of the output variables.

Corollary 1 *Let $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ be two encodings from the inputs of Algorithm 6. Then the distribution of every tuple of $(n-1)$ outputs in Algorithm 6 is independent of the distributions $x = A \times x_1 \oplus \bigoplus_{2 \le i \le n} x_i$ and $y = A \times y_1 \oplus \bigoplus_{2 \le i \le n} y_i$.*

The proof of Corollary 1 follows from that of Theorem 1 by considering output z_i. To this end, we can add at most $n-1$ indices into the set I, which correspond to $n-1$ shares of output.

The security proofs of Algorithm 4 and 5 are quite simple and we give an informal one for Algorithm 5 (masked addition) with inputs $\boldsymbol{x} = (x_1, \ldots, x_n)$ and $\boldsymbol{y} = (y_1, \ldots, y_n)$ here. The intermediate variables in the algorithm are x_i, y_i and $x_i \oplus y_i$ for $i \in \{1, \ldots, n\}$. Thus any d-family variables of above correspond to at most d shares of $\boldsymbol{x}$ or $\boldsymbol{y}$, which are uniform and independent of $\boldsymbol{x}$ or $\boldsymbol{y}$. Therefore, Algorithm 5 is secure against a d-probing adversary for $n \ge d+1$. Likewise, we can get the same conclusion for Algorithm 4.

Security of the general masking scheme. We now show the security of the general masking scheme. That is, we compose the proofs of individual masked operations to a general one, e.g., the masked (X)LS-design cipher.

Our analysis is similar to the work in [10, 2, 1]. Firstly we only provide an analysis that the composed masked operations have d-probing security. Namely, the distribution of any tuple of d or less intermediate variables in the masked cipher is independent of any plaintext or key. This requires that, for a sequence of operations, the adversary could learn d_i intermediate variables for each operation, as long as $\sum_i d_i \le d \le (n-1)/2$. We consider w masked operations $\mathcal{F} = (f_1, \ldots, f_w)$ in sequence. As shown in Figure 2, suppose that the adversary probes d_i intermediate variables in the i-th operation f_i and let ϕ_{w-1} be the input of the last operation f_w, then we can see that d_w probes of f_w are corresponding to at most $2d_w$ shares of ϕ_{w-1}. Since ϕ_{w-1} is in turn the output of f_{w-1} and by Corollary 1, the probing of d_w variables to f_w can be perfectly simulated from $2d_w$ shares of the input of f_{q-1}. By adding the probing of d_{w-1} variables of f_{w-1}, the probing of $(d_w + d_{w-1})$ variables of f_w and f_{w-1} can be perfectly simulated from $2(d_w + d_{w-1})$ shares of the input of f_{w-1}. At last, by induction we can conclude that the probing of $\sum_i d_i$ variables of the sequence of the operations can be perfectly simulated from $2\sum_i d_i$ shares of the inputs of the whole masked operations. As discussed in [1, Section 5.2], to handle the situation that adversary learns up to d variables in each execution of the masked cipher (and thus he probes many values in a multiple-run setting), the masking refreshing algorithm (i.e., Algorithm 4) should be carried out on the secret key whenever the encryption / decryption starts over again. It should be noted that

the refreshing algorithm should be called n times for nth-order masking against an $(n-1)$-probing adversary.

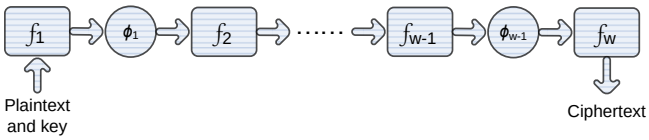

Fig. 2. The sequence of operations in consideration.

3.2 Practical evaluation

In order to illustrate Boolean matrix product masking's resistance to higher-order side-channel attacks, we evaluate the information leakage of its shares and compare it to the one of Boolean masking and inner product masking shares. We focus on the $m = 4$ case (which allows us to limit the computational cost of the evaluations) and follow the evaluation framework of [16]. Namely, we compute the mutual information between a secret m-bit value and the leakage of its n shares. For this purpose, we follow the standard simulation setting with Hamming weight power model and Gaussian noise that has been used, e.g. in [17, 1] for analyzing Boolean and inner product masking. That is, we model the leakage of shares $\boldsymbol{x} = (x_1, \ldots, x_n)$ for secret variable x as:

$$\text{Leakage}(\boldsymbol{x}) = \text{Leakage}((x_1, \ldots, x_n)) = (\text{HW}(x_1) + \epsilon_1, \ldots, \text{HW}(x_n) + \epsilon_n) \ , \quad (1)$$

where $\text{HW}(\cdot)$ denotes the Hamming weight and each ϵ_i for $i \in \{1, \ldots, n\}$ is Gaussian noise. Figure 3 shows the mutual information in $\log_{10}$ scale for the leakages of different masking schemes of order $n = 2, 3$ for 4-bit secret variables. For better comparison, we also show the leakage of inner product masking (in both $\text{GF}(2^4)$ and $\text{GF}(2^8)$) [1]. For this first experiment, we picked up a public matrix $A = (1\,1\,0\,0; 0\,0\,1\,1; 1\,0\,1\,0; 1\,1\,0\,1)$ for Boolean matrix product masking, and took $(l_1 = 1, l_2 = 15)$, $(l_1 = 1, l_2 = 255)$ and $(l_1 = 1, l_2 = 13, l_3 = 15)$ for inner product masking of order $n = 2, 3$ (similar to the choices made in [1]). Based on these settings, our observations are threefold.

First, the mutual information of all settings decreases with the noise level and both inner product and Boolean matrix product masking leak consistently less than the Boolean masking (for all noise levels).

Second, and as expected, inner product masking has lower information leakage than our masking scheme in low noise region for $n = 3$, since in our case only one share is of higher algebraic complexity. This is the price to pay for the more efficient generalization of our scheme to the higher-order cases. That is, Boolean matrix product masking should be seen as a tradeoff between Boolean masking and inner product masking in low noise contexts.

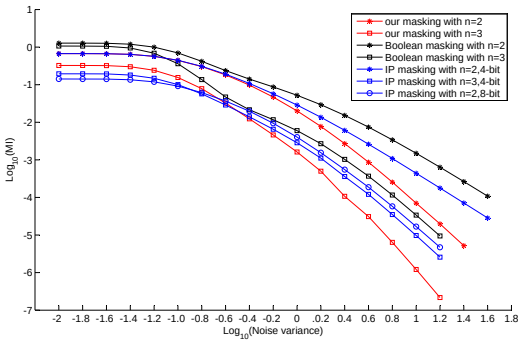

Fig. 3. Mutual information in function of the noise variance for different schemes.

Third, another important (and somewhat surprising) observation from Figure 3 is that the slopes of the IT curves for our masking scheme exceed what is predicted by the probing proof of security. For example, for $n = 2$ we can see that this slope tends to -3, while it is only -2 for Boolean and inner product masking. This is naturally a very useful observation, since it implies larger (concrete) security levels for our shares. (As proven in [5] the mutual information metric is directly proportional to the success rate of a worst-case side-channel adversary). And similar observations hold for $n = 3$. Interestingly, these reductions of the information leakages can be directly connected to the results in [9] where it is shown that the information leakages can indeed be reduced in such a way for certain types of encodings and (linear) leakage functions. However, contrary to low entropy masking schemes (which loose their security properties in case of nonlinear leakages) the "security order amplification" we observe is only a bonus in our case (i.e. even for nonlinear leakage function, we at least keep the security guarantee of the probing model). To further confirm this observation, we computed the (noise-free) statistical moments of the share's leakages for our encodings, together with the Boolean ones. Recall that in [9], the statistical moments for nth-order masking are defined as $m = \prod_{i=1}^{n}(x_i - \mathrm{E}(x_i))^{o_i}$, where E is the expectation and $o_i \in \{0, 1, 2, \ldots\}$. Thus the degree of a statistical moment is $o = \sum_{i=1}^{n} o_i$. In Table 1, the degree of the lowest key-dependent statistical moments (denoted as o_{min}) for different matrices A are listed. We can see that the value of o_{min} relates to the choice of matrix A, and specifically the minimum Hamming weight of its (or its inverse's) rows (denoted as h and ih for A and A^{-1} respectively). This can be easily explained by considering the fact that if the leakage function is linear, it will manipulate the bits of an m-bit nibble independently, and therefore the matrix multiplication has the impact of XORing more independent shares together. As a result, for a Boolean matrix product masking with n shares, the degree of the lowest key-dependent statistical moment is $o_{min} = \min(2n, h + n - 1, (n - 1) * ih + 1)$, where the $2n$ value comes from the fact that there is at least one moment of this order that is key-dependent, namely

the one multiplying the square of all the shares. For completeness, we confirm these expectations on Figure 4, where we plot the mutual information leakages of our masking scheme in function of the values of o_{min}.[5] For comparison we also add the curves of Boolean for $n = 2$.

Table 1. Degree of the lowest secret variable-dependent statistical moments.

type of masking and its degree	h, ih	o_{min}
our masking, $n = 2$	$1, 1$	2
	2,3	3
	3,4	4
	$h \geq 4$ or $ih \geq 4$	4
our masking, $n = 3$	$2, 1$	3
	2,2	4
	3,2	5
	4,2	6
	$h \geq 5$ or $ih \geq 3$	6
Boolean masking, $n = 2$		2
Boolean masking, $n = 3$		3

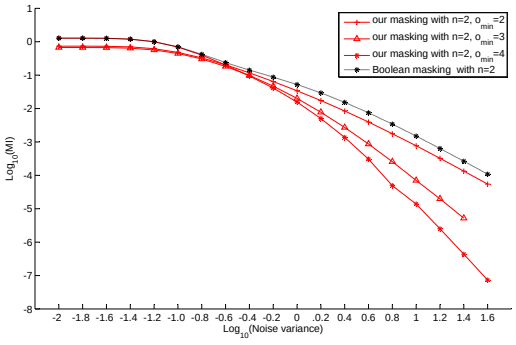

Fig. 4. Mutual information in function of the noise variance for Boolean matrix product masking using the matrices A for different degree of lowest key-dependent statistical moments.

The latter observation leads to two additional discussion points. First, by the the links between inner product masking and ours in Section 2.6, the security

[5] We take the matrices A as $(1\,0\,0\,0; 1\,1\,1\,0; 0\,0\,1\,0; 1\,1\,1\,1)$, $(1\,1\,0\,0; 0\,0\,1\,1; 1\,0\,1\,0; 1\,1\,0\,1)$ and $(1\,1\,1\,0; 1\,1\,0\,1; 1\,0\,1\,1; 0\,1\,1\,1)$ respectively.

order amplification in this section also holds for inner product masking. That is, by selecting the L vectors appropriately, we can also improve the statistical order inner product masking for linear leakage functions, which we could confirm experimentally. So by chance, the authors in [1] just picked up the worst possible L vectors for their information theoretic evaluations.

Second, as for low entropy masking schemes, such gains are not observed for non-linear leakage functions. Hence, it is an interesting scope for further research to investigate how they materialize in real-world devices. Most likely, the situation will be intermediate (i.e. lower gains than with perfectly linear leakage functions, but less informative leakages than with Boolean masking). Similarly, the order amplification observation in this section is only shown for the encoding parts of the schemes. We leave it as another interesting open question to find out if other parts of the computations maintain this property. Here as well, we conjecture that the situation will be intermediate (i.e. not all the tuples will allow order amplification for linear leakages but many of them will be less informative than with Boolean masking). So overall, this suggests inner product masking (in general) has interesting potential for reducing the number and informativeness of "tuples of interest" in masked implementations.

4 Performance Evaluation

In order to compare the efficiency of our proposed masking scheme with Boolean masking, we applied them to protect the LS-designs Fantomas [8]. The cipher uses 8-bit bitslice S-box and 16-bit L-box for 12 rounds. We implemented Boolean matrix product masking for $n = 2, 3, 4$ and $m = 4, 8$. For the part of matrix multiplication, we apply the Algorithm 9 and set the k to 1 and m for matrix of size $m \times m$ (for matrix A) and $m \times m^2$ (for matrices $\hat{A}$, $\acute{A}$ and $\grave{A}$) respectively [6]. Thus the (additional) memory for the pre-computed lookup tables is $2^m + m * 2^m * 3$ bytes, where the left side 2^m bytes come from the look-up table for $A \times x$ and the right side $m * 2^m * 3$ bytes come from the look-up takes for $\hat{A} \times x$, $\acute{A} \times x$ and $\grave{A} \times x$. As analyzed in Section 3.1, we run the masking refreshing algorithm n times on the key shares at the beginning of every execution of the cipher. We also implemented the Boolean masking [10] with the same number of shares. We wrote the codes in C language and ran them on a Atmega 2560 processor. Admittedly, the efficiency of our codes could be highly improved if rewritten in assembly language.

We summarize the performances of our implementations in Table 2. We can see that the penalty factors of our masking (using 8×8 matrix A) are not far from the ones of Boolean masking (with same n), which indicates that the efficiency of our masking scheme is comparable to Boolean masking. Note that it is counterintuitive that the performance of our masking for $m = 8$ is (slightly) better than that for $m = 4$. Yet, this is due to the fact that the smallest unit

[6] Note that in this case we don't need to keep the memory for the matrices $\hat{A}$, $\grave{A}$ and $\acute{A}$

of variable in C language is the 8-bit 'char', and thus the operations on 4×4 matrices take more time than necessary.[7]

Table 2. Performances of our implementations.

masking type	n	m	clock cycles	penalty factor
No Masking			386048	1
Boolean	2		1112064	2.88
Boolean	3		2285568	5.92
Boolean	4		3743744	9.70
Our Masking	2	8	2421760	6.27
Our Masking	3	8	4670464	12.10
Our Masking	4	8	7450624	19.30
Our Masking	2	4	2428928	6.29
Our Masking	3	4	5480448	14.1963
Our Masking	4	4	9590784	24.84

5 Conclusion

In this paper, we have proposed Boolean matrix product masking as a variant of the inner product masking in [1]. It can be used as an efficient alternative to the commonly used Boolean masking to protect bitslice ciphers such as the (X)LS-designs, and leads to efficient implementations in software computing platforms. Our scheme is proven secure in the probing model. Besides, its information theoretic analysis reveals that inner product masking can generally exhibit information leakages reduced beyond the guarantees given by the probing security order for linear leakage functions. Thanks to our matrix descriptions, we can additionally provide a simple explanation of this phenomena, which relates to the minimum Hamming weight of the multiplication matrices used in inner product masking. It is therefore an interesting scope for further research to investigate the behavior of such masking schemes in the context of concrete (close to but not exactly linear) leakage functions, and to analyze how this security order amplification be in complete implementations (and not just encodings). Incidentally, this will require the development of new analysis models and tools - since highly multivariate side-channel (e.g. information theoretic) analysis is computationally hard, and this effect is not captured by probing security.

Besides, all our analyses considered the matrix used in our multiplication as fixed and public. Yet, since this matrix is not supposed to leave the device to protect, nothing prevents the designers to keep it secret. This would make the security analysis more involved (since it would then include some kind of

[7] This problem could be solved by an optimized assembly implementation.

reverse engineering problem), but has interesting potential to further improve the security of our masking scheme without any performance penalty, which we leave as another important scope for further research.

Acknowledgements. This work has been funded in parts by the European Commission through the ERC project 280141, the CHIST-ERA project SECODE, Major State Basic Research Development Program (973 Plan) (2013CB338004). François-Xavier Standaert Standaert is a research associate of the Belgian Fund for Scientific Research (FNRS-F.R.S.). Yu Yu was supported by the National Natural Science Foundation of China Grant (Nos. 61472249, 61572192, 61572149) and International Science & Technology Cooperation & Exchange Projects of Shaanxi Province (2016KW-038). Zheng Guo was supported by the National Natural Science Foundation of China (No. 61402286) and Shanghai Minhang Innovation project (No. 2015MH069). Junrong Liu was supported by the National Natural Science Foundation of China (No. U1536103). Dawu Gu was supported by National Natural Science Foundation of China (No. 61472250).

References

1. Balasch, J., Faust, S., Gierlichs, B.: Inner product masking revisited. In: Oswald and Fischlin [12], pp. 486–510
2. Barthe, G., Belaïd, S., Dupressoir, F., Fouque, P., Grégoire, B., Strub, P.: Verified proofs of higher-order masking. In: Oswald and Fischlin [12], pp. 457–485
3. Carlet, C., Guilley, S.: Complementary dual codes for counter-measures to side-channel attacks. Adv. in Math. of Comm. 10(1), 131–150 (2016)
4. Castagnos, G., Renner, S., Zémor, G.: High-order masking by using coding theory and its application to AES. In: Cryptography and Coding - 14th IMA International Conference, IMACC 2013, Oxford, UK, December 17-19, 2013. Proceedings. pp. 193–212 (2013)
5. Duc, A., Faust, S., Standaert, F.: Making masking security proofs concrete - or how to evaluate the security of any leaking device. In: Oswald and Fischlin [12], pp. 401–429
6. Fumaroli, G., Martinelli, A., Prouff, E., Rivain, M.: Affine masking against higher-order side channel analysis. In: Selected Areas in Cryptography - 17th International Workshop, SAC 2010, Waterloo, Ontario, Canada, August 12-13, 2010, Revised Selected Papers. pp. 262–280 (2010)
7. Goubin, L., Martinelli, A.: Protecting AES with shamir's secret sharing scheme. In: Preneel and Takagi [13], pp. 79–94
8. Grosso, V., Leurent, G., Standaert, F., Varici, K.: LS-designs: Bitslice encryption for efficient masked software implementations. In: Fast Software Encryption - 21st International Workshop, FSE 2014, London, UK, March 3-5, 2014. Revised Selected Papers. pp. 18–37 (2014)
9. Grosso, V., Standaert, F., Prouff, E.: Low entropy masking schemes, revisited. In: Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers. pp. 33–43 (2013)

10. Ishai, Y., Sahai, A., Wagner, D.: Private circuits: Securing hardware against probing attacks. In: Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings. pp. 463–481 (2003)
11. Journault, A., Standaert, F.X., Varici, K.: Improving the security and efficiency of block ciphers based on LS-designs. In: 9th International Workshop on Coding and Cryptography, WCC 2015, Paris, France, April 2015 (2015)
12. Oswald, E., Fischlin, M. (eds.): Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I, Lecture Notes in Computer Science, vol. 9056. Springer (2015)
13. Preneel, B., Takagi, T. (eds.): Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings, Lecture Notes in Computer Science, vol. 6917. Springer (2011)
14. Prouff, E., Roche, T.: Higher-order glitches free implementation of the AES using secure multi-party computation protocols. In: Preneel and Takagi [13], pp. 63–78
15. Rivain, M., Prouff, E.: Provably secure higher-order masking of AES. In: Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings. pp. 413–427 (2010)
16. Standaert, F., Malkin, T., Yung, M.: A unified framework for the analysis of side-channel key recovery attacks. In: Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings. pp. 443–461 (2009)
17. Standaert, F., Veyrat-Charvillon, N., Oswald, E., Gierlichs, B., Medwed, M., Kasper, M., Mangard, S.: The world is not enough: Another look on second-order DPA. In: Abe, M. (ed.) Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6477, pp. 112–129. Springer (2010)