1

Introduction to Masking

Ange MARTINELLI and Mélissa ROSSI

Agence nationale de la sécurité des systèmes d'information, Paris, France

1.1. An overview of masking

In order to thwart the menace of passive side-channel attacks such as Differential Power Analysis (DPA) or templates attacks, the most deployed software countermeasure has been masking since its introduction in 1999. The main observation is that side-channel attacks use the relation between intermediates values and leakage traces. Thus, if intermediate values are independent from any secret, no information can be retrieved from the leakage. In a nutshell, masking consists of randomizing the intermediate data, which depend on both the secret values and known variables.

Each secret input x is basically split into $d+1$ variables $(x_i)_{0\leq i\leq d}$ referred as shares: d of them are generated uniformly at random, whereas the last one is computed such that their combination through some defined operation reveals the secret value x. The integer d is called *masking order*.

DEFINITION 1.1.– Sharing: let $\mathcal{E}$ be the finite support of the possible values for a sensitive variable x. Let $\mathsf{F} : \mathcal{E}^{d+1} \to \mathcal{E}$ be a function. A $(d+1)$-uple $\tilde{x} := (x_0, \ldots, x_d)$ is a *sharing* of x with respect to the recombination function F iff:

1) the distribution of all subsets of $(x_0, \ldots, x_d)$ of size at most d is statistically independent from x;

2) $x = \mathsf{F}(x_0, \ldots, x_d)$.

Embedded Cryptography 2,
coordinated by Emmanuel PROUFF, Guénaël RENAULT,
Matthieu RIVAIN and Colin O'FLYNN. © ISTE Ltd 2025.

The following example gives a method to design a three-sharing of an 8-bit value with respect to the XOR operation $\oplus$.

EXAMPLE 1.1.– *To mask an 8-bit value $x \in \mathbb{Z}_{2^8}$ at order 2 with respect to the recombination function $\mathsf{F}(x_0, x_1, x_2) := x_0 \oplus x_1 \oplus x_2$, we can proceed by generating two values x_0 and x_1 uniformly at random in $\mathbb{Z}_{2^8}$, and next defining $x_2 := x \oplus x_0 \oplus x_1$. That way, the second condition is automatically validated. For the first condition, any set of two values in (x_0, x_1, x_2) is indistinguishable from $\mathcal{U}(\mathbb{Z}_{2^8}) \times \mathcal{U}(\mathbb{Z}_{2^8})$ (where $\mathcal{U}$ denotes the uniform distribution in the given finite set) and the same is verified for singletons. Thus, (x_0, x_1, x_2) is a sharing of x with respect to operation $\oplus$.*

DEFINITION 1.2.– Masked scheme: let $\mathcal{C}$ be a circuit implementing a cryptographic scheme; we call *masked scheme* a scheme $\tilde{\mathcal{C}}$ that is *functionally equivalent* and where all the sensitive values are manipulated in masked form.

Note that a masked scheme is *not secure by design* and designing a masked scheme could be an error-prone task. Each masked design should be proved in a security model as will be presented later in Chapter 4 of this volume. For this purpose, we also define the notion of *gadget* that corresponds to sub-algorithms.

DEFINITION 1.3.– Gadget: a gadget is a probabilistic algorithm that takes shared and unshared inputs values and returns shared and unshared values.

In practice, while security increases with the masking order, the cost of masking can be quite heavy, both in terms of circuit size and performances and increases with the number of shares; thus, most masked scheme implementations are in first order (i.e. with two shares) or second order (i.e. with three shares).

1.2. The effect of masking on side-channel leakage

In general, if the device leaks information on one manipulated variable at a time, a d-order masking resists to stronger attacks which can combine information on at most d points of measure. In particular, Chari et al. (1999) have shown that the number of measurements required to mount a successful power analysis attack increases exponentially with the number of shares – see higher order attacks in Chapter 7 of Volume 1. Their model was considering the very classical case where each manipulated bit is leaking its noisy value, that is, the sum of its value and a Gaussian noise. Therefore, the masking order represents a trade-off between security and efficiency.

However, the reader may wonder about the precise theoretical link between masking and the potential physical leakages. Indeed, the masking countermeasure protects against an attacker able to obtain at most d intermediate values with exact precision. This attacker model is called the d-probing model. Such a model could be

far from the reality where the attacker actually obtains *many noisy intermediate values through measured leakage*. We can thus define various *leakage models* being more realistic, such as the *noisy leakage model*, or in which the security of masking is easier to assess, such as the probing model.

Let us first mention the three prominent leakage models. More details with formal definitions will be provided later in this chapter.

Probing model: in this model, besides the black-box information of the cryptosystem, the attacker is able to obtain the actual value (called "probes") of at most d exact intermediate values of its choice, where d is a parameter. Masking allows us to reach provable security in this model.

Random probing model: in this model, besides the black-box information of the cryptosystem, the attacker has access to *each intermediate value with a certain probability* p, where p is a parameter.

Noisy leakage model: in this model, besides the black-box information of the cryptosystem, the attacker has access to *all intermediate values but they are all noisy*. The standard deviation of the noise level is a parameter of this model.

The precise description and reductions between models will be detailed and proved in Chapter 4 of this volume. A high level idea that outlines the power of provable security of masking is as follows: it is possible to derive bounds on the parameters to move from a model to another. For example, if a scheme is secure in the probing model with a high d, it will be secure in the noisy leakage model with a low level of noise.

1.3. Different types of masking

As seen in our Definition 1.1, masking is a very versatile definition that can be implemented with any recombination function, purposely arbitrarily denoted as F. The choice of this function has strong repercussions on the efficiency and the leakage of the overall implementation. For efficiency matters, an operator that behaves efficiently, typically linearly, when composed with operations on shares is desirable. However, linear operation allows for easy recovery with side-channel information. In particular, the higher the algebraic complexity of the operation, the higher the impact of the noise on the recombining function, thus the higher the concrete security against side-channel attacks. In practice, masking is a costly countermeasure and efficiency matters are prominent. Thus, the vast majority of the masked schemes involve efficient recombination functions when composed of the most numerous operations of the cryptographic algorithm, often linear operations such as XOR, modular additions or multiplications by scalars. In practice, the two foremost maskings are *Boolean masking* and *arithmetic masking*.

Boolean and arithmetic masking: these families of masking provide the best trade-off concerning the efficiency and leakage. Main cryptographic operations are linear with either Boolean or arithmetic maskings. Thus, even if the leakage could be strong in case of low noise, it is sometimes less expensive to increase the masking order than to consider another type of masking with the same order.

DEFINITION 1.4.– Let $\mathcal{E}$ be the support of the possible values for a sensitive variable x. Let $\mathsf{F} : \mathcal{E}^{d+1} \to \mathcal{E}$ be a function and $q \in \mathbb{Z}$ a parameter. A Boolean, respectively, arithmetically, masked value is a sharing as defined in Definition 1.1 with $\mathsf{F}(x_0, \ldots, x_d) := x_0 \oplus \ldots \oplus x_d$, respectively, $\mathsf{F}(x_0, \ldots, x_d) := x_0 + \cdots + x_d \mod q$.

Let us remark that the space $\mathcal{E}$ is not necessarily a set of integers, and it can be any set compatible with the recombination function. Commonly, for example, a sharing can be defined over a polynomial ring like $\mathcal{E} = \mathbb{Z}_q[X]/(X^n + 1)$ for $q, n \in \mathbb{Z}$. In that case, an arithmetic masking modulus q is a natural masking solution.

Polynomial masking: this masking leverages the idea behind Lagrange polynomials: let P be a degree $d + 1$ polynomial. The knowledge of $d + 1$ couples $(y_i, \mathsf{P}(y_i))$ for any all distinct $y_0, y_1 ... y_d$ allows us to completely reconstruct the polynomial P. Conversely, the knowledge of strictly less than $d + 1$ couples $(y_i, \mathsf{P}(y_i))$ gives no information on the actual polynomial. The principle for masking a sensible value $x \in \mathcal{E}$ is simple: x is first embedded in a $(d + 1)$-degree polynomial $\mathsf{P} \in \mathcal{E}[X]$, typically x is such that $\mathsf{P}(0) = x$. The shares are therefore defined as $d + 1$ couples $x_i = (y_i, \mathsf{P}(y_i))$, and we choose here $(1, \mathsf{P}(1))$, $(2, \mathsf{P}(2)), \ldots, (d + 1, \mathsf{P}(d + 1))$. To recover the polynomial with the knowledge of all the $\mathsf{P}(i)$, we can recover the polynomial through Lagrange's formula:

$$L_{\mathsf{P}(1),\ldots,\mathsf{P}(d+1)}(X) := \sum_{i=1}^{d+1} \mathsf{P}(i) \cdot \left(\prod_{j=1, j\neq i}^{j=d+1} \frac{X - j}{i - j} \right)$$

Next, we can recombine the secret with $\mathsf{F}(x_0, \ldots, x_d) = L_{\mathsf{P}(1),\ldots,\mathsf{P}(d+1)}(0)$.

This type of masking has a quite complex recombination function, thus it intuitively leads to a very low practical leakage while being somehow more expensive in terms of performance. Indeed, few common cryptographic operations are linear with a decomposition on polynomial images.

Multiplicative masking: the multiplicative masking consists of applying $\mathsf{F}(x_0, \ldots, x_d) := x_0 \times \cdots \times x_d$ in Definition 1.1, where the multiplication is made in the corresponding field $\mathcal{E}$. This type of masking can be very well suited for the SubBytes step of the AES. Indeed, $x \mapsto x^{-1}$ is an endomorphism of $GF(256)^*$.

Practical activity: Algorithm 1.1 is applying this technique for a field inversion after an addition of the key to the plaintext. The inversion is performed in $GF(256)^*$ and it is sequentially applied to every byte of the 128-bit state, denoted as S. For the sake of simplicity and because the XOR operation is outside of the scope of this activity, the masked state $S = (S_0, \ldots, S_d)$ given as input of the algorithm is the masked XOR of the key k and the plaintext P. The shares $S_1, \ldots S_d$ are generated uniformly at random in $(GF(256)^*)^{16}$ and $S_0 = x \times S_1 \times \cdots \times S_d$. In the additional material, some traces of this algorithm are provided with the corresponding plaintexts. They were derived on a ChipWhisperer-Lite Xmega. A high-order side-channel attack seems tedious; you could exploit a property of multiplicative masking to recover the whole secret k with a simple first-order DPA (see Chapter 4 of Volume 1).

Algorithm 1.1.

Secret : A multiplicative masking $(k_0, \cdots k_4) \in [0, 255]^5$ of a key $k = k_0 \cdot k_1 \cdots k_4 \in [0, 255]$
Input : A multiplicative masking $(S_0, \cdots S_4) \in ([0, 255]^{16})^5$ of a value $P \oplus k$ for a known P

```
for b ∈ [0, 15] do
    for i ∈ [0, 4] do
        S_i[b] := (S_i[b])^254
    end
end
```

Inner-product masking and affine masking: as outlined above, multiplicative masking has some vulnerabilities if not used properly but we can combine it with a Boolean one as follows.

DEFINITION 1.5.– Let $x \in \mathcal{E}$ be a sensitive value, $(v_0, \ldots, v_d) \in \mathcal{E}$ be fixed vector and $\langle \cdot, \cdot \rangle$ be a $\oplus$-linear inner product defined on $\mathcal{E}$. An inner product masked value is a masked value as defined in Definition 1.1, with:

$$\mathsf{F}(x_0, \ldots, x_d) := \langle (x_0, \ldots, x_d), (v_0, \ldots, v_d) \rangle.$$

The particular case of $d = 1$ and $v_1 = 1$ is called an *affine masking*.

This masking is slightly less efficient than the Boolean masking because of the extra multiplication operations but it may offer stronger security as the leakage can be better de-correlated from the secret. This kind of masking and the following schemes aim at lowering down the amount of information which can be recovered from the leakage. They can all offer a better security/complexity tradeoff in a specific setup.

Leakage squeezing: this consists of applying a different bijection to each share and proceeds with standard masking – mostly Boolean. The squeezing functions need

to be bijective to recover the plain shares during computation and to unmask the output. Eventually, first-order Boolean masking combined with leakage squeezing gives similar security than masking with two to three more shares for a different complexity, more suited in hardware.

DEFINITION 1.6.– Let $x = (x_0, \ldots, x_d) \in \mathcal{E}^{d+1}$ be a masked sensitive value. Let $(B_0, \ldots, B_d)$ be a set of bijections over $\mathcal{E}$. We call leakage squeezing the application of the B_is over x such that the operated circuit works on $\tilde{x} = (B_0(x_0), \ldots, B_d(x_d))$.

Orthogonal direct sum masking: in order to ensure dual security against both passive and active attacks, masking using error correcting codes has been proposed as follows.

DEFINITION 1.7.– Let $x \in \mathcal{E}$ be a sensitive value and $\mathbf{y} \in \mathcal{E}^d$ a vector of masks. Let G and H be generator matrices of two linear codes $\mathcal{C}$ and $\mathcal{D}$ such that $\mathcal{C} \cap \mathcal{D} = \{0\}$ and $\mathcal{C} + \mathcal{D} = \mathcal{E}^{d+1}$. We can define the orthogonal direct sum masking of x as $z = xG \oplus \mathbf{y}H$, where xG denote the scalar–vector product and $\mathbf{y}H$ the vector–matrix product. We can then recover x as $x = zG^T(GG^T)^{-1}$.

1.4. Code-based masking: toward a generic framework

While Boolean and arithmetic masking are the most common choices, a lot of other operations are possible as outlined in section 1.4. Many more masking recombination functions F have been attempted to improve either the security in some settings or to provide a more global theoretical background. In this section, we outline a generalization of the Boolean masking family. Indeed, the Boolean masking can be seen as a sub-case of the inner product masking. This can go further: inner product masking is a sub-case of leakage squeezing (LS), itself generalized as orthogonal direct sum masking. As a side methodology with some intersections, there stands polynomial masking.

As an attempt at uniformization, generalized code-based masking has been introduced to give a common framework to all of these schemes as shown in Figure 1.1. It has been shown that any other masking scheme outlined in section 1.4 can be instantiated using generalized code-based masking.

Generalized code-based masking is similar to orthogonal direct sum masking where $\mathcal{C}$ and $\mathcal{D}$ can be any matrices such that $dim(\mathcal{C}) + dim(\mathcal{D}) \leq d + 1$. Let us define the matrices G and H as generator matrices of the linear codes $\mathcal{C}$ and $\mathcal{D}$, respectively. Let k be the dimension of G and t the dimension of H. Table 1.1 gives an overview of how to parameterize code-based masking to instantiate specific masking.

Scheme	Boolean	IP	LS	ODS	Polynomial	GCB
Conditions	$\mathcal{C} \cap \mathcal{D} = \{0\}$ $\mathcal{C} + \mathcal{D} = \mathcal{E}^{d+1}$	$\mathcal{C} \cap \mathcal{D} = \{0\}$ $\mathcal{C} + \mathcal{D} = \mathcal{E}^{d+1}$	$\mathcal{C} \cap \mathcal{D} = \{0\}$ $\mathcal{C} + \mathcal{D} = \mathcal{E}^{d+1}$	$\mathcal{C} \cap \mathcal{D} = \{0\}$	$\mathcal{C} \cap \mathcal{D} = \{0\}$	$\mathcal{C} \cap \mathcal{D} = \{0\}$
$\boldsymbol{G \in \mathcal{E}^{k \times d}}$	$(1\,0\,0 \cdots 0)$	$(1\,0\,0 \cdots 0)$	$G \in \mathcal{E}^{k \times d}$	$G \in \mathcal{E}^{k \times d}$	$(1\,1 \cdots 1)$	$G \in \mathcal{E}^{k \times d}$
$\boldsymbol{H \in \mathcal{E}^{t \times d}}$	$\begin{pmatrix} 1\,0 \cdots 0 \\ 1\,1 \cdots 0 \\ \vdots\ \vdots\ \ddots\ \vdots \\ 1\,0 \cdots 1 \end{pmatrix}$	$\begin{pmatrix} a_1\ 1\ 0 \cdots 0 \\ a_2\ 0\ 1 \cdots 0 \\ \vdots\ \vdots\ \vdots\ \ddots\ \vdots \\ a_t\ 0\ 0 \cdots 1 \end{pmatrix}$	$H \in \mathcal{E}^{t \times d}$	$H \in \mathcal{E}^{t \times d}$	$\begin{pmatrix} a_1^1\ a_2^1 \cdots a_d^1 \\ a_1^2\ a_2^2 \cdots a_d^2 \\ \vdots\ \vdots\ \ddots\ \vdots \\ a_1^t\ a_2^t \cdots a_d^t \end{pmatrix}$	$H \in \mathcal{E}^{t \times d}$
Parameters	$k = 1, d = t + 1$	$k = 1, d = t + 1$	$d = k + t$	$d = k + t$	$d \geq k + t$	$d \geq k + t$

Table 1.1. *Instantiation of masking schemes through generalized code-based masking*

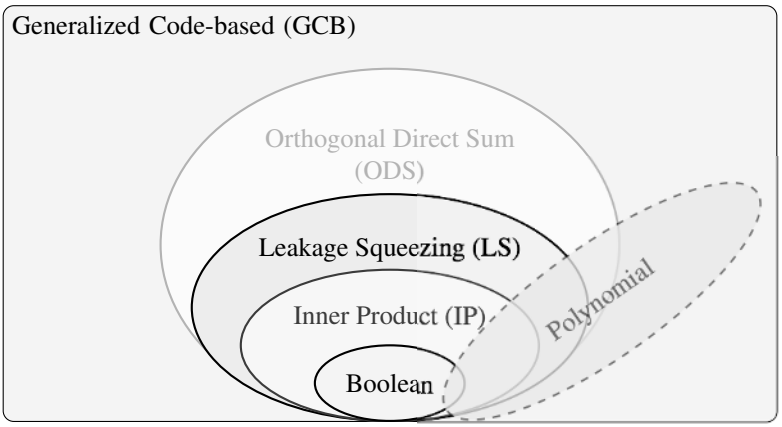

Figure 1.1. *Generalization of masking schemes under code-based masking. For a color version of this figure, see www.iste.co.uk/prouff/cryptography2.zip*

1.5. Hybrid masking

One type of masking is not often suited for a whole cryptographic algorithm. For example, whereas a multiplicative masking can be well suited for a SubBytes application corresponding to $x \mapsto x^{254}$ in $GF(256)$, it is not suited for the addition of the sub-keys in the value (this time linear for $\oplus$). Furthermore, an arithmetic masking could be perfectly suited for addition of polynomials in $\mathbb{Z}_q[X]/(X^n+1)$ for $q, n \in \mathbb{Z}$, but it is not well suited for deriving a masked outcome of a comparison $|p| \geq K$ for p being a polynomial coefficient and K being a constant. These cases are very frequent in cryptographic algorithms. Hence, several types of masking can be necessary for an efficient masked algorithm. In consequence, *conversion algorithms are necessary*. The particularity of these algorithms is the fact that they are functionally equivalent to the identity function; however, they take a masked variable as input and return this same variable but masked with a different type of masking.

There is no generic technique for deriving such algorithms independently of the types of masking. Efficient conversions are sometimes challenging to design in a secure way. The case of Boolean to arithmetic conversions will be depicted later in Chapter 2 of this volume. There also exist less widespread conversions like multiplicative to Boolean. Some additional information on these conversions will be pointed out later in the references.

1.6. Examples of specific maskings

Masking is a very powerful countermeasure and is usually implemented either table-based or directly implementing masked operations. Nevertheless, some algorithms can be efficiently masked with specific methods.

As will be detailed in Chapter 9 of this volume, the RSA algorithm has simple and efficient ways to do exponent *blinding* to protect the exponentiation modulo, the product of two primes. This kind of masking uses a random multiple of the order of the multiplicative group to protect the exponentiation. The eventual output stays unchanged but the actual exponent used in the computation is randomized.

Mini-game: let us try to wear the adversary's shoes. Here are several masked algorithms, and we assume that they are functionally correct, that is, the outputs are correctly computed as functions of the inputs. However, for each algorithm, there is a masking flaw in the probing model (informally defined earlier in this chapter and formally defined later in Chapter 4 of this volume). In a masking context, the goal of an adversary against the masking is to break Definition 1.1. Thus, the goal is to find a subset of less than d variables, which is correlated to the secret.

1) Here, $d = 2$. This algorithm is simply computing an XOR of its inputs. The secret here is a.

Algorithm 1.2.

Input : Boolean masking (a_0, a_1), $(b_0, b_1) \in [0, 255]^2$
Output: Boolean masking $(c_0, c_1) \in [0, 255]^2$

1 $c_0 := a_0 \oplus a_1$
2 $c_0 := c_0 \oplus b_0$
3 $c_1 := b_1$
4 **return** (c_0, c_1)

2) Here, $d = 3$. This algorithm is computing the multiplication of two inputs masked in Boolean form. Your goal is to recover a with at most two intermediate values.

3) In this algorithm, $d = 4$. This algorithm generates an arithmetic masking in $\mathbb{Z}/2^{16}\mathbb{Z}$ of a sample whose distribution is centered and has a small standard deviation. Let us define D as a centered discrete Gaussian distribution on $\mathbb{Z}/2^{16}\mathbb{Z}$ with a small standard deviation 0.5. Note that distribution does not correspond to the final one, and the precise description of the final distribution is not necessary for finding flaws in this algorithm. Your goal is to find three intermediate variables that provide some information about the secret, the output c.

Algorithm 1.3.

Input : $(a_0, a_1, a_2), (b_0, b_1, b_2) \in [0, 255]^3$
Output: $(c_0, c_1, c_2) \in [0, 255]^3$

$r_{0,1}, r_{0,2}, r_{1,2} \xleftarrow{\$} [0, 255]$
$r_{1,0} := a_0 b_1 \oplus a_2 b_1$
$r_{2,0} := (r_{0,2} \oplus a_0 b_2) \oplus a_1 b_0$
$r_{2,1} := (r_{0,1} \oplus a_1 b_2) \oplus a_2 b_0$
$c_0 := a_0 b_0 \oplus (r_{0,1} \oplus r_{0,2})$
$c_1 := a_1 b_1 \oplus r_{1,0}$
$c_2 := a_2 b_2 \oplus (r_{2,0} \oplus r_{2,1})$
return (c_0, c_1, c_2)

Algorithm 1.4.

Output: $(c_0, c_1, c_2, c_3) \in \mathbb{Z}/2^{16}\mathbb{Z}$
for $i \in [0, 3]$ **do**
  $r_i \leftarrow D$
  $y_i \xleftarrow{\$} \mathbb{Z}/2^{16}\mathbb{Z}$`/* drawing in` $\mathbb{Z}/2^{16}\mathbb{Z}$ `uniformly at random */`
  $c_i := r_i + y_i$
end
return (c_0, c_1, c_2, c_3)

1.7. Outline of the part

Masking is a very powerful tool to generically protect cryptographic algorithms against side-channel attacks. However, there is an important tradeoff between efficiency and security with many levels, and it will be the main subject of this part.

In Chapter 2, several common cryptographic algorithms and functions will be masked with emphasis on elementary operations.

Chapter 3 of this volume outlines the impact of hardware on masking and the necessity to take into account physical phenomena like glitches in the design of masking implementations. Chapter 4 focuses on the various security models available to prove masking implementations. Eventually, Chapter 5 gives tools to implement masking schemes on specific algorithms and to assess their security.

1.8. Notes and further references

– Section 1.1. The theory of masking has been seminally introduced in Goubin and Patarin (1999); Ishai et al. (2003); Goudarzi and Rivain (2016).

– Section 1.2. For more information about the effect of masking on side-channel leakage, we refer to Chari et al. (1999) and Barthe et al. (2017), and more information on the different models will be presented later in the chapter.

– Section 1.3. For concrete examples of Boolean, arithmetic and multiplicative masking, we refer to Rivain and Prouff (2010) and Genelle et al. (2011).

– Section 1.4. For more information and recent works in the domain of code-based masking, we recommand Bringer et al. (2014); Wang et al. (2020) and Cheng et al. (2021).

1.9. References

Barthe, G., Dupressoir, F., Faust, S., Grégoire, B., Standaert, F.-X., Strub, P.-Y. (2017). Parallel implementations of masking schemes and the bounded moment leakage model. Report 2016/912, Cryptology ePrint Archive [Online]. Available at: https://eprint.iacr.org/2016/912.

Bringer, J., Carlet, C., Chabanne, H., Guilley, S., Maghrebi, H. (2014). Orthogonal direct sum masking: A smartcard friendly computation paradigm in a code, with builtin protection against side-channel and fault attacks. In *WISTP*. Springer, Berlin, Heidelberg.

Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P. (1999). Towards sound approaches to counteract power-analysis attacks. In *Advances in Cryptology – CRYPTO' 99*, Wiener, M. (ed.). Springer, Berlin, Heidelberg.

Cheng, W., Guilley, S., Carlet, C., Danger, J.-L., Mesnager, S. (2021). Information leakages in code-based masking: A unified quantification approach. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 3, 465–495.

Genelle, L., Prouff, E., Quisquater, M. (2011). Thwarting higher-order side channel analysis with additive and multiplicative maskings. In *Cryptographic Hardware and Embedded Systems – CHES 2011*, Preneel, B. and Takagi, T. (eds). Springer, Berlin, Heidelberg.

Goubin, L. and Patarin, J. (1999). DES and differential power analysis (the "duplication" method). In *Cryptographic Hardware and Embedded Systems – CHES 1999*, Koç, Ç.K. and Paar, C. (eds). Springer, Berlin, Heidelberg.

Goudarzi, D. and Rivain, M. (2016). On the multiplicative complexity of Boolean functions and bitsliced higher-order masking. In *Cryptographic Hardware and Embedded Systems – CHES 2016*. Springer, Berlin, Heidelberg.

Ishai, Y., Sahai, A., Wagner, D. (2003). Private circuits: Securing hardware against probing attacks. In *Advances in Cryptology – CRYPTO 2003*, Boneh, D. (ed.). Springer, Berlin, Heidelberg.

Rivain, M. and Prouff, E. (2010). Provably secure higher-order masking of AES. In *Cryptographic Hardware and Embedded Systems – CHES 2010*, Mangard, S. and Standaert, F.X. (eds). Springer, Berlin, Heidelberg.

Wang, W., Méaux, P., Cassiers, G., Standaert, F.-X. (2020). Efficient and private computations with code-based masking. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2, 128–171.