A New Generic Fault Resistant Masking Scheme using Error-Correcting Codes

Chloé Gravouil[1,2]

[1] Univ Rennes, CNRS, IRMAR - UMR 6625, F-35000 Rennes, France, chloe.gravouil@univ-rennes1.fr
[2] EDSI, 8 rue du Bordage, 35510 Cesson-Sévigné, France, chloe.gravouil@nagra.fr

Keywords: Masking Scheme · Fault Resistance · Error Correcting Codes

Abstract. One of the main security challenges white-box cryptography needs to address is side-channel security. To this end, designers aim to eliminate the dependence between variables and sensitive data. Classical countermeasures to do so are masking schemes. Nevertheless, most masking schemes are not designed to thwart the other main security threat : fault attacks. Thus, we aimed to build a masking scheme that could combine resistance to both of these types of attacks.
In this paper, we present our new generic fault resistant masking scheme using BCH error-correcting codes, as well as the design choices behind it.

1 Introduction

Over the course of the last twenty years, devices like smartphones or IoT have taken a growing part in the daily life of billions of persons. More and more applications have been designed to bring a great diversity of functionalities to users, nevertheless this development has come with a counterpart : it makes those devices a valuable target to attackers. This is why many open devices applications need to implement cryptography.

Thus, a new field of cryptography has emerged, named white-box cryptography, in opposition to black-box cryptography where the attacker has only access to inputs and outputs of the implementation, and grey-box cryptography, where the attacker can moreover exploit flaws of the implementation. In white-box cryptography, the attacker has a total access over the execution platform of the algorithm and its implementation. Therefore, this model matches the best with the realistic case, where the attacker can even be the owner of the device.

The side-channel attacks are one of the major types of attacks, inherited from the grey-box cryptography model, that a white-box cryptography designer needs to thwart. Those types of attacks rely on flaws of the implementation, that are correlations between sensitive data in the algorithm, for instance keys, and physical data leakage during an execution of the implementation. Those data leakage can be of different types, like execution time, electromagnetic emanations, or power consumption of the device executing the implementation [Cad05].

Thereby, side-channel countermeasures aim to eliminate any relation between sensitive data and those physical data leakages [PR13]. For example, in the timing attack case, conditionals statements depending on the sensitive data must be avoided. However, the most commonly performed type of side-channel attacks is the power-monitoring attack. Among the variety of countermeasures to this type of attacks, masking is the most developed topic, that relies on the following basis :

In a field $\mathbb{K}$, a variable X is split into n_{in} sub-variables $X_0, ..., X_{n_{in}-1}$, named *shares*, such that $X = X_0 \oplus ... \oplus X_{n_{in}-1}$, and each tuple of at most $(n_{in} - 1)$ variables X_i is independent from X, i.e.

$$\begin{gathered}\forall i \in \{0, ..., n_{in} - 1\}, \forall v \in \mathbb{K}, \forall (v_0, .., v_{i-1}, v_{i+1}, .., v_{n_{in}-1}) \in \mathbb{K}^{n_{in}-1}, \\ P((X_0, .., X_{i-1}, X_{i+1}, .., X_{n_{in}-1}) = (v_0, .., v_{i-1}, v_{i+1}, .., v_{n_{in}-1}) | X = v) = \\ P((X_0, .., X_{i-1}, X_{i+1}, .., X_{n_{in}-1}) = (v_0, .., v_{i-1}, v_{i+1}, .., v_{n_{in}-1}))\end{gathered}$$

To that end, the values of the variables $X_0, \ldots, X_{n_{in}-2}$ are chosen uniformly at random, and the value of $X_{n_{in}-1}$ is computed so that $X = X_0 \oplus \ldots \oplus X_{n_{in}-1}$. Therefore, if a value Z is correlated to a sensitive value X, then, as each share Z_i separately is independent from Z, they are independent from the sensitive value X. As stated in [BBP+16], the tuple $(Z_i)_{0 \leq i \leq n_{in}-1}$ still depends on X, but because of the noise, the complexity of extracting information is exponential in the number of shares n_{in}.

Each set of n_{in} sub-variables X_i such that $X = X_0 \oplus \ldots \oplus X_{n_{in}-1}$ is named an n_{in}-sharing of X. *Masking schemes* describe how, for a given function f with n inputs and m outputs, the sharings of the outputs are built as functions of the input shares.

Indeed, for f a function with n inputs $X_0, \ldots, X_{n-1}$ and one output Y, a (n_{in}, n_{out})-masking scheme F of the function f such that $(Y_0, \ldots, Y_{n_{out}-1}) = F(X_0, \ldots, X_{n_{in}-1})$ verify that :

$$\begin{aligned}
Y_0 &= F_0(X_{0,0}, \ldots, X_{0,n_{in}-1}, \ldots, X_{n-1,0}, \ldots, X_{n-1,n_{in}-1}) \\
Y_1 &= F_1(X_{0,0}, \ldots, X_{0,n_{in}-1}, \ldots, X_{n-1,0}, \ldots, X_{n-1,n_{in}-1}) \\
&\vdots \\
Y_{n_{out}-2} &= F_{n_{out}-2}(X_{0,0}, \ldots, X_{0,n_{in}-1}, \ldots, X_{n-1,0}, \ldots, X_{n-1,n_{in}-1}) \\
Y_{n_{out}-1} &= F_{n_{out}-1}(X_{0,0}, \ldots, X_{0,n_{in}-1}, \ldots, X_{n-1,0}, \ldots, X_{n-1,n_{in}-1})
\end{aligned},$$

with $X_i = \sum_{j=0}^{n_{in}-1} X_{i,j} \, \forall i \in \{0, \ldots, n-1\}$ and $Y = \sum_{j=0}^{n_{out}-1} Y_j$.

However, most classic masking schemes do not offer resistance against fault attacks ([ISW03],[BDF+17],[BBP+16]). Fault attacks consists of disrupting the correct functioning of the cryptographical primitive to observe faulty behaviour of variables depending on sensitive data ([GT04], [Ott05]). The leaked information can then be processed with statistic or analytic methods, thereby disclosing all or part of sensitive data involved in the computation. This is why we aimed

to develop a new masking scheme that would resist to those attacks, hence protecting against side-channel and fault attacks.

Outline of the paper : In Section 2, we introduce the notations that will be used in the rest of the paper, then the masking schemes and error correcting code properties that will be used thereafter. We justify our design choices in Section 3 and describe the masking scheme in Section 4. Finally, in Section 5, we present the performances of our scheme applied to an AES bitsliced implementation.

2 Preliminaries

In this paper we will use the following notations :

- For a random variable X, we note $P(X)$ its *probability distribution* and H its *entropy* ($H(X) = -\sum_x P(X = x) \log_2(P(X = x))$)
- For random variables X and Y,
  - We note $H(X|Y) = \sum_x \sum_y P(X = x, Y = y) \log \frac{P(X=x,Y=y)}{P(Y=y)}$ the *conditional entropy* of X given Y.
  - $I(X;Y) = H(X) - H(X|Y)$ is the *mutual information* between X and Y.
- We consider $\mathbb{K}$ a field with $char(\mathbb{K}) = 2$.
- For an array $A \in \mathbb{K}^n$, we note $HW(A)$ the Hamming weight of A, i.e. the number of non-zero elements of A.

2.1 Masking Schemes

Many properties have been introduced in the state-of-the-art to quantify the security of circuits. In 1999, Chari et al. introduced in their paper [CJRR99] the noisy leakage model which aims to be the more realistic leakage model, where the adversary can obtain leaked values that are sampled thanks to a Gaussian distribution centered on the real value of the sensitive variables. This model was extended later by Rivain and Prouff in [PR13] to general noise distributions. The noisy leakage model allows to simulate leakage in a rather precise manner, but is not very easy to use in practice. That is why, in [ISW03], Ishai et al. introduced the d-probing security model.

Property 1 (d-probing security). A circuit is *d-probing secure* if and only if every set of d intermediate variables is independent of any sensitive variable x. For every set of d probes $(q_0, \ldots, q_{d-1})$, we have

$$I\left(q_0 \cup \ldots \cup q_{d-1}; x\right) = 0$$

This property is much easier to prove than security in the noisy leakage model, but was thought to be not sufficiently accurate to describe the leakage. However, in 2014, Duc et al. proved in [DDF14] that security in the d-probing model implies security in the noisy leakage model.

Different implementation properties have been proven to be sufficient conditions for probing security when joined.

Correctness The first property that all masking schemes must follow is correctness ([Bil15]). This property doesn't bring security by itself, but is needed for obvious reasons of keeping the functionality of the function to mask f. Indeed, for a masking F of a function f such that $Y = f(X_1, \ldots, X_n)$, we need to have

$$\bigoplus_{i=0}^{n_{out}-1} F_i\left((X_{0,j})_{0\leq j\leq n_{in}-1}, ..., (X_{n-1,j})_{0\leq j\leq n_{in}-1}\right) = f\left(\bigoplus_{j=0}^{n_{in}-1} X_{0,j}, ..., \bigoplus_{j=0}^{n_{in}-1} X_{n-1,j}\right)$$

Non-Completeness

Property 2 (Non-Completeness, [NRR06], [Bil15]). A masking scheme F verifies *non-completeness* if every component function F_i is independent of at least one share of each of the input variables.

Intuitively, non-completeness is necessary supposing that a probe on combinational block (i.e. an operation) implies the leakage of all inputs to the combinational block ([RBN+15]). Thus, for a combinational block scheme that does not satisfy non-complete-ness, a probe would imply the leakage of all shares by at least one input variable.

Uniformity

Property 3 (Uniform Masking, [Bil15]). A masking scheme F of a function $f : \mathbb{K}^n \to \mathbb{K}$ with n inputs X_j and one output Y is said to be *uniform* if and only if

$$\forall (x_0, \ldots, x_{n-1}) \in \mathbb{K}^n, \forall (y_0, \ldots, y_{n_{out}-1}) \in \mathbb{K}^{n_{out}},$$
$$P\left((Y_j)_{0\leq j\leq n_{out}-1} = (y_j)_{0\leq j\leq n_{out}-1} \middle| (X_j)_{0\leq j\leq n-1} = (x_j)_{0\leq j\leq n-1}\right) = \begin{cases} \frac{1}{|\mathbb{K}|^{n_{out}-1}} & \text{if } f(x_0, \ldots, x_{n-1}) = \bigoplus_{j=0}^{n_{out}-1} y_j \\ 0 & \text{otherwise} \end{cases}$$

In other words, uniformity implies that for each n-tuple of input values $(x_0, \ldots, x_{n-1})$, all n_{out}-sharings $(Y_0, \ldots, Y_{n_{out}-1})$ computed by F with a n_{in}-sharing of $(x_0, \ldots, x_{n-1})$ as input are equiprobable.

Non-Completeness and Uniformity imply 1-probing security In [NRR06], Nikova et al. prove that security of implementations against first-order attacks relies on correctness, non-completeness and uniformity.

Lemma 1 ([NRR06]). *Non-completeness and uniformity implies 1-glitch probing extended security.*

State of the art of masking schemes As our purpose was to develop an AND masking scheme itself composed of boolean operations, we tested the properties introduced above on ISW ([ISW03]) and the three different boolean multiplication gadgets presented in [GJRS18] :

- The BDF+ algorithm of [BDF+17]
- The BBP+ algorithm of [BBP+16]
- The BCPZ algorithm of [BCPZ16]

The results are presented below.

	ISW			BDF+			BBP+			BCPZ		
Number of shares d	2	4	8	2	4	8	2	4	8	2	4	8
Uniformity	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
1^{st}-Order Non-Completeness	✗	✗	✗	✗	✓	✓	✗	✗	✗	✗	✗	✗
2^{nd}-Order Non-Completeness	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Probing Security Order	1	3	7	1	3	7	1	1	3	1	3	7

Table 1: Properties of ISW, BDF+, BBP+ and BCPZ gadgets

2.2 BCH error correcting codes

For our masking scheme described in section 4, we aimed to use an error correcting code with an easy management of codewords parities, hence the choice of cyclic codes. In particular, we choose BCH codes as it can be decoded in constant time via the Peterson-Gorenstein-Zierler algorithm ([Pet60]).

Definition 1 (Cyclic codes [ABO09]). *Let $n \in \mathbb{N}^\star$. A linear code $\mathcal{C}$ of length n is said to be a **cyclic code** if all cyclic permutations of codewords belong to the code as well, i.e., for example*

$$(c_0, c_1, ..., c_{n-1}) \in \mathcal{C} \Longrightarrow (c_{n-1}, c_0, c_1, ..., c_{n-2}) \in \mathcal{C}$$

Property 4. For a cyclic code $\mathcal{C}$ of length n over a finite field $\mathbb{F}_q$,

- we can identify each $(c_0, c_1, ..., c_{n-1}) \in \mathbb{F}_q^n$ to $c_0 + c_1X + c_2X^2 + ... + c_{n-1}X^{n-1} \in \mathbb{F}_q[X]/(X^n - 1)$ and reciprocally.
- there exists a polynomial $g(X) \in \mathbb{F}_q[X]$ such that for each $(c_0, c_1, ..., c_{n-1}) \in \mathcal{C}$, $g(X)|c_0 + c_1X + c_2X^2 + ... + c_{n-1}X^{n-1}$.

BCH error correcting codes are based on the following mathematical notions and properties. Thereafter, we will assume q to be a prime power.

Definition 2. *Let $n \in \mathbb{N}^\star$ such that $n \wedge q = 1$, m the multiplicative order of q modulo n. Let $s \in \{0, ..., n-1\}$. We note $C(s)$ the q-cyclotomic class of s modulo n*

$$C(s) = \{s, s * q, ..., s * q^{m_s - 1}\},$$
with m_s *the smallest non-zero integer such that* $s = s * q^{m_s} \pmod{n}$.

Definition 3. *Let* $n \in \mathbb{N}^\star$ *such that* $n \wedge q = 1$, m *the multiplicative order of* q *modulo* n *and* α *an element of* $\mathbb{F}_{q^m}$ *of multiplicative order* n. *Let* $s \in \{0, ..., n-1\}$. *We note*

$$M_{\alpha^s}(X) = \prod_{i \in C(s)} (X - \alpha^i) = \prod_{i=0}^{m_s - 1} (X - \alpha^{sq^i}).$$

Property 5. $M_{\alpha^s} \in \mathbb{F}_q[X]$.

Property 6. M_{α^s} is irreducible over $\mathbb{F}_q$.

Subsequently, we can define BCH (Bose Chaudhuri Hocquenghem) error correcting codes as follows :

Definition 4 (BCH Code, [BRC60b],[BRC60a]). *Let* $n \in \mathbb{N}^\star$, m *the multiplicative order of* q *modulo* n, α *an element of* $\mathbb{F}_{q^m}$ *of multiplicative order* n *and* $b, \delta \in \mathbb{N}$ *such that* $\delta \geq 3$. *A cyclic code of length* n *over* $\mathbb{F}_q$ *is said to be a* ***BCH code*** *of minimum Hamming distance at least* δ *if its generator polynomial* g *verifies*

$$g(X) = lcm(M_{\alpha^b}(X), M_{\alpha^{b+1}}(X), ..., M_{\alpha^{b+\delta-2}}(X))$$

Property 7 (Correction Capacity). Let $\mathcal{C}$ be an error correcting code of minimum Hamming distance d. We define the correction capacity t of the code to be $\lfloor \frac{d-1}{2} \rfloor$. The correction capacity of a BCH code of minimum Hamming distance at least δ verifies $t \geq \lfloor \frac{d-1}{2} \rfloor$.

3 Design Rationale

3.1 Masking Scheme Design Constraints

We aim that our scheme verify the three implementation properties listed in Lemma 1 to ensure first order probing security. We want to build a scheme to compute $a \wedge b$, with $a, b \in \mathbb{F}_2$, while supposing that all input and output shares are codewords. We also want to introduce randomness with random polynomials R_i of respective parities r_i.

To achieve this, we want to build variables $s_i \in \mathbb{F}_2$ as parities of the output shares S_i of our scheme. Therefore, they would depend on products between parities a_i, b_i of input shares, products $a_i r_j$ or $r_i b_j$ between parities of input shares and parities of random polynomials, and products $r_i r_j$ between parities of random polynomials. Finally, by the correctness property, the sum of these variables s_i would be equal to $a \wedge b$.

Non-Completeness implies a condition on the number of shares First of all, to be able to perform one instance of the scheme after another, the number of output shares needs to be equal to the number of shares of each input. Therefore, to comply with non-completeness, the number of input shares of both a and b cannot be equal to 2. Indeed, in this case, we suppose that

- a is represented by two codewords shares A_0 and A_1 of respective parities a_0 and a_1 such that $a_0 \oplus a_1 = a$.
- b is represented by two codewords shares B_0 and B_1 of respective parities b_0 and b_1 such that $b_0 \oplus b_1 = b$.
- There are two output codewords shares S_0 and S_1 of respective parities s_0 and s_1 depending on parities a_0, a_1, b_0 and b_1, such that $s_0 \oplus s_1 = a \wedge b$.

As $s_0 \oplus s_1 = a \wedge b = a_0b_0 \oplus a_0b_1 \oplus a_1b_0 \oplus a_1b_1$, there are four different products a_ib_j to distribute among two variables s_i, so at least two in each variable s_i. But the sum of any two of these products does not verify non-completeness. Hence, the number n_{in} of shares of each input and of the output of the scheme needs to be at least 3.

Randomness Requirement and its impact on the BCH code choice To keep a reasonable randomness requirement of our scheme, we imposed ourselves that the number of random variables associated to each input would be stricly less than the number of shares of each input. For instance, the number of random polynomials parities r_i multiplied to the parities a_i is strictly less than the number of B_i shares, and reciprocally.

We note n_r the number of random polynomials involved in the scheme. We suppose the number of random polynomials associated to each input to be equal, i.e. there are as many random parities r_j that are to be multiplied to input shares parities a_i than random parities r_i to be multiplied to input shares parities b_j. Hence, n_r is even.

As the number n_{in} of input shares A_i is equal to the number of input shares B_i, and we just supposed that the number of random polynomials associated to each input is equal to $\frac{n_r}{2}$, it implies that $\frac{n_r}{2} < n_{in}$.

Hence, if we suppose $n_{in} = 3$ we can have either $\frac{n_r}{2} = 1$ or 2. It implies that there are at most either $(3+1)^2$ or $(3+2)^2$ products of variables potentially involved in the computation of s_i variables. The number of these products is an upper bound of the length of the BCH code used in the scheme, as codewords serving as masks will be applied to the array gathering them (see section 4). In the same manner, n_{in}^2 is a lower bound of the length code as it corresponds to the number of products a_ib_j, imperatively involved in the computations of s_i variables for correctness reasons.

In subsection 3.2, we will detail why a BCH code of correction capacity at least 2 is needed. Moreover, as we consider a_i and b_j the parities of codewords, the generator polynomial needs to be of odd parity so that variables a_i and b_j could take either values in $\mathbb{F}_2$. For $n_{in} = 3$ and thus a code length n with $9 \leq n \leq 25$, the maximum potential dimension of the code is 2^{12}. Furthermore,

we knew that subsequently we would need codewords to act as masks to be applied to a matrix of cross-products, in order to compute output parities s_i. As all cross-products involved of form $a_i r_j$ or $r_i b_j$ appear in an even number of variables s_i, those codewords would need to have a certain number of exponents in common. To that end, we considered a search space of such cardinality to be too small to ensure that we would find n_{in} codewords suitable to the s_i computation formulas we would determine thereafter.

If the number of shares n_{in} is 4, the code length n verifies $16 \leq n \leq 49$. Retaining the code properties listed above, we selected between the potential code length values the one that would maximize the dimension of the corresponding BCH code. In this case, the maximum dimension value possible is 2^{29}, when $n = 45$.

That is why we picked the number of shares $n_{in} = 4$ and the number of random polynomials associated to each input $\frac{n_r}{2} = 3$. Indeed, if $\frac{n_r}{2} \leq 2$, $(n_{in} + \frac{n_r}{2})^2 < n = 45$, hence a contradiction to the fact that $(n_{in} + \frac{n_r}{2})^2$ is an upper bound of the value of n. Subsequently, the BCH code used in the scheme has length $n = 45$, dimension 2^{29} and generator polynomial $g(X) = lcm(M_\alpha(X), M_{\alpha^2}(X), M_{\alpha^3}(X), M_{\alpha^4}(X)) = (X^{12} + X^3 + 1) * (X^4 + X + 1) = X^{16} + X^{13} + X^{12} + X^7 + X^3 + X + 1$.

Ensuring Uniformity and Correctness As we chose the number of shares to be $n_{in} = 4$ and the number of random polynomials associated to each input $\frac{n_r}{2} = 3$, there are at most 49 different products of parities involved in the computation of the parity values s_i of the output share S_i :

- 16 products of form $a_i b_j$ that need to be present in an odd number of s_i computations.
- 12 products of form $a_i r_j$, 12 products of form $r_i b_j$ and 9 products of form $r_i r_j$ that need to be present in an even number of s_i computations.

We noticed that for uniformity reasons, it is compulsory that each s_i includes at least one single parity (a_i, b_i or r_i). Indeed, as variables a_i, b_i and r_i are independent and equiprobable, all products of two of those variables have $(\frac{3}{4}, \frac{1}{4})$ as probability vector. Hence, sums of those products cannot be equiprobable. That is why we add single a_i, b_i or r_i to the potential operands of variables s_i.

We first randomly split the 16 products of form $a_i b_j$ between s_0, s_1, s_2 and s_3 so that they still comply with non-completeness. At this point, each s_i variable is not equiprobable, so we first started to add three single variables to pairs of s_i to ensure global equiprobability, and equiprobability conditionned by the value of $a \wedge b$ necessary for the uniformity property.

We then added products of form $a_i r_j$,$r_i b_j$ or $r_i r_j$ one after another to pairs of s_i variables, so that, when conditionned by either value of $a \wedge b$, the probability of each possible value of (s_0, s_1, s_2, s_3) tends to $\left(\frac{1}{2}\right)^3$ until reaching it, while retaining non-completeness. We obtained the following formulas :

- $s_0 = r_1 b_1 \oplus r_2 b_1 \oplus r_0 \oplus a_1 r_3 \oplus a_2 r_5 \oplus a_2 b_2 \oplus a_3 r_3 \oplus a_3 b_2 \oplus r_1 b_0 \oplus a_3 b_1 \oplus r_1 r_5 \oplus r_2 r_5$

- $s_1 = r_1b_1 \oplus r_0 \oplus a_2b_1 \oplus a_0b_2 \oplus a_2r_5 \oplus a_0b_1 \oplus a_0r_3 \oplus a_3b_0 \oplus r_2b_2 \oplus a_0b_0 \oplus r_2r_5 \oplus a_2 \oplus r_1r_5 \oplus r_0r_3$
- $s_2 = a_3b_3 \oplus a_1r_5 \oplus r_5 \oplus a_3r_3 \oplus r_2b_2 \oplus a_1b_2 \oplus r_1b_0 \oplus r_2r_5 \oplus r_2b_0 \oplus a_2 \oplus r_0r_3 \oplus a_2r_4$
- $s_3 = a_2b_3 \oplus r_2b_1 \oplus a_1r_5 \oplus a_1b_1 \oplus r_5 \oplus a_1r_3 \oplus a_0r_3 \oplus a_1b_0 \oplus a_0b_3 \oplus r_2r_5 \oplus r_2b_0 \oplus a_1b_3 \oplus a_2b_0 \oplus a_2r_4$

3.2 Input Shares Correction Design

The aim of the correcting design is to be able to correct the AND inputs, i.e. correct faults committed at the end of the preceding operations or just before the beginning of the current AND. That is why we suppose all input and output shares of our scheme to be codewords.

To minimize the number of corrections needed, we compute and correct sub-sums of input shares instead of correcting each share. For non-completeness compliance, it is not possible to add all shares A_i of A (or all shares B_i of B) in a same sub-sum.

Furthermore, we aimed to use the smallest possible number of sub-sums while computing them so that, even if they are sums of shares of both A and B, it is feasible in case of a fault to determine if it was committed on A or B. To that end, we choose the following sub-sums V_0, V_1 and V_2 and correct them into respective codewords noted V_0', V_1' and V_2':

- $V_0 = A_0 \oplus A_1 \oplus B_0 \oplus B_1$
- $V_1 = A_2 \oplus A_3 \oplus B_2 \oplus B_3$
- $V_2 = A_2 \oplus A_3 \oplus B_0 \oplus B_1$

Thus, V_0 and V_1 cover all possible input shares faults spots, while, in case of a fault detected by one of those two subsums, V_2 allows to determine whether the fault has been committed on a share of A or a share of B, as detailed below.

- $A_0' = A_0 \oplus V_0$
- $A_1' = A_1 \oplus V_1'$
- $A_2' = A_2 \oplus V_2$
- $A_3' = A_3 \oplus V_2'$

- $B_0' = B_0 \oplus V_1$
- $B_1' = B_1 \oplus V_0'$
- $B_2' = B_2 \oplus V_2$
- $B_3' = B_3 \oplus V_2'$

We can first notice that introducing a one bit fault on share A_0 yields the same impact as introducing a fault A_1, as for introducing a fault on A_2 or A_3, on B_0 or B_1, or on B_2 or B_3. Hence, to simplify, we will only consider faults in A_0, A_3, B_0 or B_3 below.

Table 2 represents the correctness of parities of variables V_i depending on fault location, i.e. if a fault on one input share (A_0, A_3, B_0 or B_3) implies a fault on the parity of the sub-sums V_0, V_1 or V_2. Table 3 represents the correctness of parities of input shares after the XOR of variables V_i and V_i', i.e. if , after the xor of sub-sums V_i and their corrections V_i', each modified input share A_i' or B_i' carries the same parity as would do the corresponding input share in a non-faulted environment.

Fault Location \ V_i	V_0	V_1	V_2
A_0	✗	✓	✓
A_3	✓	✗	✗
B_0	✗	✓	✗
B_3	✓	✗	✓

Table 2: Correctness of parity of variables V_i depending on input share single fault location

Fault Location \ Modified Input Share	A'_0	A'_1	A'_2	A'_3	B'_0	B'_1	B'_2	B'_3
A_0	✓	✓	✓	✓	✓	✓	✓	✓
A_3	✓	✓	✗	✗	✗	✓	✗	✓
B_0	✗	✓	✗	✓	✗	✓	✗	✓
B_3	✓	✓	✓	✓	✗	✓	✓	✗

Table 3: Correctness of parities of input shares A'_i and B'_i after the XOR of variables V_i and V'_i in a single fault location case

Hence, it can be noticed that the overall parities of $A_0 \oplus A_1 \oplus A_2 \oplus A_3$ and $B_0 \oplus B_1 \oplus B_2 \oplus B_3$ are exact after correction. We can see that they also remain exact after introducing a fault in two different input shares, regardless of the bit indexes of these two faults.

Fault Locations \ V_i		V_0	V_1	V_2
A_0	A_2	✗	✗	✗
A_0	B_0	✓	✓	✗
A_0	B_2	✗	✗	✓
A_2	B_0	✗	✗	✓
A_2	B_2	✓	✓	✗
B_0	B_2	✗	✗	✗

Table 4: Correctness of parity of variables V_i depending on two input share fault locations

Fault Locations	Modified Input Share	A'_0	A'_1	A'_2	A'_3	B'_0	B'_1	B'_2	B'_3
A_0	A_2	✓	✓	✓	✓	✗	✓	✗	✓
A_0	B_0	✗	✓	✗	✗	✗	✓	✗	✓
A_0	B_2	✗	✓	✗	✓	✗	✓	✗	✓
A_2	B_0	✓	✓	✓	✓	✗	✓	✓	✗
A_2	B_2	✗	✓	✗	✓	✗	✓	✗	✓
B_0	B_2	✓	✓	✓	✓	✗	✓	✓	✗

Table 5: Correctness of parities of input shares A_i and B_i after the XOR of variables V_i and V'_i in a two faults locations case

Thus, it can be noticed that using only two variables V_i would not be enough for the correction of parities of input shares. That is why the minimum number of subsums needed to cover all spots while complying with non-completeness is three. Furthermore, in the two faults scenario, each subsum will at most carry two bit faults. In this case, the subsum, despite being faulted, represents the correct parity, as parities are computed modulo 2. We nevertheless imposed that the used BCH code would have a correction capacity of 2, to avoid that a correction operation of two faults for a BCH code of correction capacity strictly less than two could modify the subsum in question and its then-correct parity.

Finally, at this point, not all $A_0, A_1, A_2, A_3, B_0, B_1, B_2$ and B_3 are necessarily codewords anymore (in the case where a fault has been detected). Nevertheless, they have the correct parity, which is the only requirement for the following step.

4 Our Masking Scheme

As explained in section 3.1, we choose to use the BCH code length $n = 45$. We then considered $m = 12$ the multiplicative order of 2 modulo n and α be a 45^{th} primitive root of unity. Thus, we subsequently use the BCH code of generator polynomial $g(X) = lcm(M_\alpha(X), M_{\alpha^2}(X), M_{\alpha^3}(X), M_{\alpha^4}(X))=M_\alpha(X) * M_{\alpha^3}(X) = (X^{12}+X^3+1)*(X^4+X+1) = X^{16}+X^{13}+X^{12}+X^7+X^3+X+1$.

We represent $a, b \in \mathbb{F}_2$ by codewords $A, B \in \mathbb{F}_2^{45}$ such that $HW(A) \bmod 2 = a$ and $HW(B) \bmod 2 = b$. We suppose each of those two input codewords to be split between four shares, i.e. $A = A_0 \oplus A_1 \oplus A_2 \oplus A_3$ and $B = B_0 \oplus B_1 \oplus B_2 \oplus B_3$, with the A_i and B_i being codewords as well.

We first perform correction on the input shares to prevent any faults introduced in the end of preceding operations or just before the start of the current one. To avoid the correction of the eight shares, we compute three intermediate sub-sums V_0, V_1 and V_2 and correct them into respective codewords V'_0, V'_1 and V'_2:

- $V_0 = A_0 \oplus A_1 \oplus B_0 \oplus B_1$

- $V_1 = A_2 \oplus A_3 \oplus B_2 \oplus B_3$
- $V_2 = A_2 \oplus A_3 \oplus B_0 \oplus B_1$

Thus, thanks to these variables V_i and their corresponding corrected codewords V_i', we can correct the parities of shares $A_0, A_1, A_2, A_3, B_0, B_1, B_2$ and B_3 as follows :

- $A_0 = A_0 \oplus V_0$
- $A_1 = A_1 \oplus V_1'$
- $A_2 = A_2 \oplus V_2$
- $A_3 = A_3 \oplus V_2'$
- $B_0 = B_0 \oplus V_1$
- $B_1 = B_1 \oplus V_0'$
- $B_2 = B_2 \oplus V_2$
- $B_3 = B_3 \oplus V_2'$

We consider $n_r = 6$ random polynomials $R_0, R_1, R_2, R_3, R_4, R_5 \in \mathbb{F}_2^{45}$. Then, noting for each of those polynomials X_i that $x_i = HW(X_i) \bmod 2$, we compute the following array :

$$mCP = \begin{bmatrix} r_2b_3 & a_3b_3 & r_1b_1 & a_2b_3 & r_2b_1 & a_3r_4 & r_0 & a_1r_5 & a_1b_1 \\ a_0r_4 & a_2b_1 & r_0b_2 & a_0b_2 & r_5 & r_1r_3 & a_1r_3 & a_1r_4 & a_2r_5 \\ a_2b_2 & a_3r_3 & a_0b_1 & a_0r_3 & a_3b_0 & r_0b_1 & a_1b_0 & r_0r_5 & r_2r_3 \\ r_1r_4 & a_3b_2 & r_2b_2 & a_1b_2 & a_0b_0 & r_1b_0 & r_0b_0 & a_3b_1 & a_0b_3 \\ a_0r_5 & r_2r_5 & r_2b_0 & a_2 & r_1r_5 & a_1b_3 & r_0r_3 & a_2b_0 & a_2r_4 \end{bmatrix}$$

The positioning of the products in the array enables to compute the output shares parities s_i (section 3.1) by applying to mCP the following masks $maskS_0, maskS_1, maskS_2, maskS_3 \in \mathbb{F}_2^{45}$ that are also codewords. Indeed, each parity s_i verifies $s_i = \sum_{j=0}^{44} (mCP[j] \ \& \ maskS_i[j])$.

Hence, they could be corrected a few times among all the masked AND occurrences of an implementation.

$$maskS_0 = \begin{bmatrix} 0&0&1&0&1&0&1&0&0 \\ 0&0&0&0&0&0&1&0&1 \\ 1&1&0&0&0&0&0&0&0 \\ 0&1&0&0&0&1&0&1&0 \\ 0&1&0&0&1&0&0&0&0 \end{bmatrix} \iff \begin{aligned} ms_0(X) &= X^{42} + X^{40} + X^{38} + X^{29} \\ &+ X^{27} + X^{26} + X^{25} + X^{16} \\ &+ X^{12} + X^{10} + X^7 + X^4, \\ &\textit{with } ms_0 = 0 \textit{ mod } g(X) \end{aligned}$$

$$maskS_1 = \begin{bmatrix} 0&0&1&0&0&0&1&0&0 \\ 0&1&0&1&0&0&0&0&1 \\ 0&0&1&1&1&0&0&0&0 \\ 0&0&1&0&1&0&0&0&0 \\ 0&1&0&1&1&0&1&0&0 \end{bmatrix} \iff \begin{aligned} ms_1(X) &= X^{42} + X^{38} + X^{34} + X^{32} \\ &+ X^{27} + X^{24} + X^{23} + X^{22} \\ &+ X^{15} + X^{13} + X^7 + X^5 \\ &+ X^4 + X^2, \\ &\textit{with } ms_1 = 0 \textit{ mod } g(X) \end{aligned}$$

$$maskS_2 = \begin{bmatrix} 0\,1\,0\,0\,0\,0\,0\,1\,0 \\ 0\,0\,0\,0\,1\,0\,0\,0\,0 \\ 0\,1\,0\,0\,0\,0\,0\,0\,0 \\ 0\,0\,1\,1\,0\,1\,0\,0\,0 \\ 0\,1\,1\,1\,0\,0\,1\,0\,1 \end{bmatrix} \iff \begin{aligned} ms_2(X) &= X^{43} + X^{37} + X^{31} + X^{25} \\ &+ X^{15} + X^{14} + X^{12} + X^7 \\ &+ X^6 + X^5 + X^2 + 1, \\ &with\ ms_2 = 0\ mod\ g(X) \end{aligned}$$

$$maskS_3 = \begin{bmatrix} 0\,0\,0\,1\,1\,0\,0\,1\,1 \\ 0\,0\,0\,0\,1\,0\,1\,0\,0 \\ 0\,0\,0\,1\,0\,0\,1\,0\,0 \\ 0\,0\,0\,0\,0\,0\,0\,0\,1 \\ 0\,1\,1\,0\,0\,1\,0\,1\,1 \end{bmatrix} \iff \begin{aligned} ms_3(X) &= X^{41} + X^{40} + X^{37} + X^{36} \\ &+ X^{31} + X^{29} + X^{23} + X^{20} \\ &+ X^9 + X^7 + X^6 + X^3 \\ &+ X + 1, \\ &with\ ms_3 = 0\ mod\ g(X) \end{aligned}$$

For $0 \leq i \leq 3$, we compute the variables $x_{i,0}, x_{i,1}, x_{i,2}, x_{i,3} \in \mathbb{F}_2$ depending on variables a_i, b_i and r_i such that respectively

- $s_0, s_1, s_2, x_{0,0}, x_{0,1}, x_{0,2}$ and $x_{0,3}$
- $s_1, s_0, s_3, x_{1,0}, x_{1,1}, x_{1,2}$ and $x_{1,3}$
- $s_2, s_0, s_3, x_{2,0}, x_{2,1}, x_{2,2}$ and $x_{2,3}$
- $s_3, s_1, s_2, x_{3,0}, x_{3,1}, x_{3,2}$ and $x_{3,3}$

are sets of independant and equiprobable variables.

Then, we randomly chose four codewords $C_{0,0}, C_{1,0}, C_{2,0}, C_{3,0} \in \mathbb{F}_2^{45}$ of odd parity, and twenty-four of even parity ($C_{0,j}, C_{1,j}, C_{2,j}, C_{3,j} \in \mathbb{F}_2^{45}$ for $1 \leq j \leq 6$).

Hence, we compute output shares $S_0, S_1, S_2, S_3 \in \mathbb{F}_2^{45}$ such that

- $S_0 = s_0{*}C_{0,0}{+}s_1{*}C_{0,1}{+}s_2{*}C_{0,2}{+}x_{0,0}{*}C_{0,3}{+}x_{0,1}{*}C_{0,4}{+}x_{0,2}{*}C_{0,5}{+}x_{0,3}{*}C_{0,6}$
- $S_1 = s_1{*}C_{1,0}{+}s_0{*}C_{1,1}{+}s_3{*}C_{1,2}{+}x_{1,0}{*}C_{1,3}{+}x_{1,1}{*}C_{1,4}{+}x_{1,2}{*}C_{1,5}{+}x_{1,3}{*}C_{1,6}$
- $S_2 = s_2{*}C_{2,0}{+}s_0{*}C_{2,1}{+}s_3{*}C_{2,2}{+}x_{2,0}{*}C_{2,3}{+}x_{2,1}{*}C_{2,4}{+}x_{2,2}{*}C_{2,5}{+}x_{2,3}{*}C_{2,6}$
- $S_3 = s_3{*}C_{3,0}{+}s_1{*}C_{3,1}{+}s_2{*}C_{3,2}{+}x_{3,0}{*}C_{3,3}{+}x_{3,1}{*}C_{3,4}{+}x_{3,2}{*}C_{3,5}{+}x_{3,3}{*}C_{3,6}$

In this manner, each output share S_i can equiprobably take 2^7 different values. With the codewords $C_{i,j}$ we chose, we obtained

$$\begin{aligned} S_0 = (&s_0, s_1 \oplus x_{0,0}, s_2, s_0 \oplus x_{0,1}, s_0 \oplus x_{0,0}, x_{0,2} \oplus x_{0,0}, x_{0,2}, s_1, s_1 \oplus s_2 \oplus x_{0,2} \\ &\oplus x_{0,1}, s_2, x_{0,2}, s_1 \oplus x_{0,1}, s_2, s_1, s_1 \oplus s_2 \oplus x_{0,3} \oplus x_{0,1}, s_2, s_1, s_1 \oplus s_2 \\ &\oplus x_{0,2}, s_2, x_{0,3}, x_{0,3} \oplus x_{0,1}, x_{0,2}, s_1, s_2, s_0 \oplus x_{0,1}, s_0, s_1, s_2 \oplus x_{0,1}, \\ &x_{0,2}, s_0, x_{0,3} \oplus x_{0,1}, s_0 \oplus x_{0,0}, x_{0,3} \oplus x_{0,1}, s_0, x_{0,2}, x_{0,2} \oplus x_{0,0}, x_{0,2} \oplus x_{0,1}, \\ &x_{0,3} \oplus x_{0,0}, s_0 \oplus x_{0,0}, x_{0,3}, s_0 \oplus x_{0,0}, s_0, x_{0,3}, x_{0,3} \oplus x_{0,0}, x_{0,3} \oplus x_{0,0}) \end{aligned}$$

$$
\begin{aligned}
S_1 = (&s_0, s_1, s_3 \oplus x_{1,0}, s_0, s_1, s_1, s_0, x_{1,3} \oplus x_{1,1}, s_3 \oplus x_{1,0}, s_3 \oplus x_{1,1}, x_{1,3}, x_{1,3} \\
&\oplus x_{1,0}, s_3, x_{1,3} \oplus x_{1,1}, s_3 \oplus x_{1,1}, s_3 \oplus s_0 \oplus x_{1,1} \oplus x_{1,1}, s_0 \oplus x_{1,0}, s_3 \oplus \\
&x_{1,1} \oplus x_{1,3}, s_3, x_{1,3}, x_{1,3} \oplus x_{1,0}, s_0, s_1 \oplus x_{1,1}, s_3 \oplus x_{1,1} \oplus x_{1,0}, s_0, s_0, \\
&s_1 \oplus x_{1,1}, s_3 \oplus s_0 \oplus x_{1,1}, s_0, s_1, x_{1,1}, s_1 \oplus x_{1,1}, s_1, x_{1,1}, s_1 \oplus x_{1,0}, x_{1,1}, \\
&x_{1,1} \oplus x_{1,1}, s_1, s_1 \oplus x_{1,0}, x_{1,1}, x_{1,3} \oplus x_{1,0}, x_{1,1} \oplus x_{1,0}, x_{1,1}, x_{1,3}, x_{1,3}) \\
S_2 = (&s_3, x_{2,0} \oplus x_{2,2}, s_3, x_{2,3} \oplus x_{2,1}, x_{2,0} \oplus x_{2,1}, x_{2,0} \oplus x_{2,2}, s_2, s_2 \oplus x_{2,2}, s_2 \\
&\oplus x_{2,2}, x_{2,3}, s_0, s_3 \oplus x_{2,1}, s_3 \oplus s_0 \oplus x_{2,3}, s_0 \oplus x_{2,1} \oplus x_{2,2}, s_3, s_2, s_0 \oplus \\
&x_{2,1} \oplus x_{2,2}, x_{2,3} \oplus x_{2,2}, s_2, s_2, s_2 \oplus x_{2,1}, x_{2,3} \oplus x_{2,1}, s_3, s_0, x_{2,3}, s_2, s_2, \\
&s_3 \oplus x_{2,1}, s_2, s_2, s_0, s_3 \oplus s_0 \oplus x_{2,0} \oplus x_{2,2}, s_0 \oplus x_{2,1}, s_3, s_3 \oplus x_{2,2}, s_0 \oplus \\
&x_{2,3} \oplus x_{2,0} \oplus x_{2,1} \oplus x_{2,2}, s_0, x_{2,0}, x_{2,0}, x_{2,3}, x_{2,0}, x_{2,3}, x_{2,3}, x_{2,0}, x_{2,0}) \\
S_3 = (&x_{3,2}, s_2, s_1, x_{3,2}, x_{3,2}, s_2, x_{3,2} \oplus x_{3,1}, s_2, s_2 \oplus s_1, x_{3,2} \oplus x_{3,1}, x_{3,2} \oplus x_{3,1} \\
&\oplus x_{3,0}, s_1, s_3 \oplus x_{3,0}, s_2 \oplus x_{3,0}, x_{3,3}, s_3, s_2 \oplus s_1 \oplus x_{3,0}, s_2 \oplus x_{3,3}, x_{3,3} \\
&\oplus x_{3,1}, s_3, s_1 \oplus x_{3,3}, s_3 \oplus x_{3,1}, s_3, s_1 \oplus x_{3,0}, s_3, s_3, x_{3,3}, x_{3,3} \oplus x_{3,1}, s_3 \\
&\oplus x_{3,1}, x_{3,3}, x_{3,3} \oplus x_{3,0}, s_2 \oplus x_{3,0}, x_{3,3} \oplus x_{3,0}, s_3 \oplus x_{3,1}, s_2 \oplus s_1 \oplus x_{3,1}, \\
&s_2 \oplus x_{3,0}, x_{3,2} \oplus x_{3,0}, s_3, s_1, s_3, s_1 \oplus x_{3,2}, s_1, x_{3,2} \oplus x_{3,1}, x_{3,2}, x_{3,3})
\end{aligned}
$$

5 Application to a global implemention

To be able to apply the masking scheme presented above on a bitsliced implementation of any cryptographic primitive, another boolean operation needs to be addressed : the NOT operation. As a matter of fact, the XOR operation is linear, hence the input shares of a XOR can be just XORed to one another to compute output shares, and the OR operation is a combination of an AND operation and three NOT operations.

5.1 Implementation of NOT operation

As we consider input and output shares of the AND operation to be codewords, this needs to still be the case for the NOT operation for compatibility reasons.

Hence, the NOT operations takes as input shares codewords A_0, A_1, A_2 and A_3 and returns codewords B_0, B_1, B_2 and B_3 such that $HW(A_0) + HW(A_1) + HW(A_2) + HW(A_3) \mod 2 = HW(B_0) + HW(B_1) + HW(B_2) + HW(B_3) + 1 \mod 2$.

The idea of the implementation is to add random codewords to the three first input shares A_0, A_1 and A_2, then add to the fourth (A_3) a codeword of parity opposite to the parity of the sum of the three first random codewords.

To that end, we choose three random messages m_0, m_1 and m_2 and use them such that :

- $B_0 = A_0 \oplus (m_0(X) * g(X))$

- $B_1 = A_1 \oplus (m_1(X) * g(X))$
- $B_2 = A_2 \oplus (m_2(X) * g(X))$
- $B_3 = A_3 \oplus ((m_0(X) + m_1(X) + m_2(X)) * g(X) \oplus g(X))$

Thereby $A_0 \oplus A_1 \oplus A_2 \oplus A_3 \oplus B_0 \oplus B_1 \oplus B_2 \oplus B_3 = g(X)$ and $g(X)$ has been chosen to have odd parity. Thus $HW(B_0 \oplus B_1 \oplus B_2 \oplus B_3) \mod 2 = HW(A_0 \oplus A_1 \oplus A_2 \oplus A_3) + 1 \mod 2$.

5.2 Tests

We tested this masking scheme on a AES implementation with a randomly-chosen fixed key using the TBoxes of [CEJv03] bitsliced with the Usuba tool ([Mer20]). This implementation is composed of 37586 AND gates, 2293 NOT gates and 66751 XOR gates.

We tested with a processor Intel(R) Core(TM) i7-6600U CPU @ 2.60GHz three different versions of this implementation :

- The raw bitsliced implementation where the TBoxes are bitsliced and ShiftRows and MixColumn are implemented with boolean operations AND, OR, NOT and XOR as well.
- The bitsliced implementation where the AND operation is masked by the ISW masking scheme ([ISW03]), the NOT operation is performed by flipping one bit share and the XOR operation is implemented by XORing shares.
- The bitsliced implementation where the AND operation is masked by the masking scheme presented in this paper, the NOT operation is performed as described in subsection 5.1 and the XOR operation is implemented by XORing polynomial shares.

We obtained the following values:

	Raw Bitsliced Implementation	Implementation Masked With ISW [ISW03]	Implementation Masked With Our Scheme
Time for 1000 executions	0.29 s	1103.12 s	2123.46 s
Binary Size	2.3 MB	3.6 MB	3.8 MB

6 Conclusion

In this paper, we present the AND masking scheme we built in order to combine the natural goal of masking schemes, that is to say side-channel resistance, and the resistance to fault attacks. To do so, we used error correcting codewords shares and designed our scheme to be compliant with uniformity, non-completeness and correctness properties.

The decreasing performances are due to the fact that each bit is represented by $4 * 45$ bits. To mitigate the loss, we consider the possibility of adapting the

scheme to different number of shares, and to do so to use BCH codes of non-maximal cardinality, or to use error correcting codes other than BCH codes, in particular with power of 2 lengths to ease implementation.

For further work, different leads are open. First, the uniformity, non-completeness and correctness properties ensure first-order probing security, so we would need to determine the exact probing security order of the scheme. Furthermore, we would like to investigate the balance between the performances gain and the correction loss for an implementation where not all masked AND include the correction design.

References

ABO09. Daniel Augot, Emanuele Betti, and Emmanuela Orsini. An introduction to linear and cyclic codes. In Massimiliano Sala, Shojiro Sakata, Teo Mora, Carlo Traverso, and Ludovic Perret, editors, *Gröbner Bases, Coding, and Cryptography*, pages 47–68. Springer, 2009.

BBP+16. Sonia Belaïd, Fabrice Benhamouda, Alain Passelègue, Emmanuel Prouff, Adrian Thillard, and Damien Vergnaud. Randomness complexity of private circuits for multiplication. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 616–648. Springer, Heidelberg, May 2016.

BCPZ16. Alberto Battistello, Jean-Sébastien Coron, Emmanuel Prouff, and Rina Zeitoun. Horizontal side-channel attacks and countermeasures on the ISW masking scheme. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *CHES 2016*, volume 9813 of *LNCS*, pages 23–39. Springer, Heidelberg, August 2016.

BDF+17. Gilles Barthe, François Dupressoir, Sebastian Faust, Benjamin Grégoire, François-Xavier Standaert, and Pierre-Yves Strub. Parallel implementations of masking schemes and the bounded moment leakage model. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 535–566. Springer, Heidelberg, April / May 2017.

Bil15. Begül Bilgin. *Threshold implementations : as countermeasure against higher-order differential power analysis*. PhD thesis, University of Twente, Enschede, Netherlands, 2015.

BRC60a. R.C. Bose and D.K. Ray-Chaudhuri. Further results on error correcting binary group codes. *Information and Control*, 3(3):279–290, 1960.

BRC60b. R.C. Bose and D.K. Ray-Chaudhuri. On a class of error correcting binary group codes. *Information and Control*, 3(1):68–79, 1960.

Cad05. Tom Caddy. Side-channel attacks. In Henk C. A. van Tilborg, editor, *Encyclopedia of Cryptography and Security*. Springer, 2005.

CEJv03. Stanley Chow, Philip A. Eisen, Harold Johnson, and Paul C. van Oorschot. White-box cryptography and an AES implementation. In Kaisa Nyberg and Howard M. Heys, editors, *SAC 2002*, volume 2595 of *LNCS*, pages 250–270. Springer, Heidelberg, August 2003.

CJRR99. Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 398–412. Springer, Heidelberg, August 1999.

DDF14. Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models: from probing attacks to noisy leakage. Cryptology ePrint Archive, Report 2014/079, 2014. https://eprint.iacr.org/2014/079.

GJRS18. Dahmun Goudarzi, Anthony Journault, Matthieu Rivain, and François-Xavier Standaert. Secure multiplication for bitslice higher-order masking: Optimisation and comparison. In Junfeng Fan and Benedikt Gierlichs, editors, *COSADE 2018*, volume 10815 of *LNCS*, pages 3–22. Springer, Heidelberg, April 2018.

GT04. Christophe Giraud and Hugues Thiebeauld. A survey on fault attacks. In Jean-Jacques Quisquater, Pierre Paradinas, Yves Deswarte, and Anas Abou El Kalam, editors, *Smart Card Research and Advanced Applications VI, IFIP 18th World Computer Congress, TC8/WG8.8 & TC11/WG11.2 Sixth International Conference on Smart Card Research and Advanced Applications (CARDIS), 22-27 August 2004, Toulouse, France*, volume 153 of *IFIP*, pages 159–176. Kluwer/Springer, 2004.

ISW03. Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer, Heidelberg, August 2003.

Mer20. Darius Mercadier. *Usuba, Optimizing Bitslicing Compiler. (Usuba, Compilateur Bitslicing Optimisant).* PhD thesis, Sorbonne University, France, 2020.

NRR06. Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold implementations against side-channel attacks and glitches. In Peng Ning, Sihan Qing, and Ninghui Li, editors, *ICICS 06*, volume 4307 of *LNCS*, pages 529–545. Springer, Heidelberg, December 2006.

Ott05. Martin Otto. *Fault attacks and countermeasures.* PhD thesis, University of Paderborn, Germany, 2005.

Pet60. W. Wesley Peterson. Encoding and error-correction procedures for the bose-chaudhuri codes. *IRE Trans. Inf. Theory*, 6(4):459–470, 1960.

PR13. Emmanuel Prouff and Matthieu Rivain. Masking against side-channel attacks: A formal security proof. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 142–159. Springer, Heidelberg, May 2013.

RBN+15. Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating masking schemes. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 764–783. Springer, Heidelberg, August 2015.