Formal Evaluation and Construction of Glitch-resistant Masked Functions

Sofiane Takarabt, Sylvain Guilley, Youssef Souissi, Khaled Karray, Laurent Sauvage, Yves Mathieu

Formal Evaluation and Construction of Glitch-resistant Masked Functions

1st Sofiane Takarabt
Secure-IC
Paris, France
forename.name@secure-ic.com

2nd Sylvain Guilley
Secure-IC
Paris, France
forename.name@secure-ic.com

3rd Youssef Souissi
Secure-IC
Paris, France
forename.name@secure-ic.com

4th Khaled Karray
Secure-IC
Paris, France
forename.name@secure-ic.com

5th Laurent Sauvage
LTCI, Télécom Paris
Palaiseau, France
forename.name@telecom-paris.com

6th Yves Mathieu
LTCI, Télécom Paris
Palaiseau, France
forename.name@telecom-paris.com

***Abstract*—We give an algorithm that checks whether every possible transition is masked. It allows to verify the absence of first-order leakage from a masked netlist. It validates the state-of-the-art masking schemes, such as Threshold Implementation and Domain Oriented Masking, but also proves that more compact netlists with equivalent functions are secure. We leverage this methodology to propose a more compact implementation of AES S-Box.**

***Index Terms*—Cryptography, Side-channel attacks, Data protection, Masking countermeasure, Glitches, Secure S-Box implementation.**

I. Introduction

During the last two decades, the Side-Channel Attack (SCA) [16], [17] was one of the most important challenges in cryptographic implementations. Since unprotected implementations can leak sensitive data through power consumption and Electromagnetic Emanation (EM) [7], protecting against SCA becomes a fundamental task to make implementations more secure. To do so, designers proposed some countermeasures by randomizing and hiding the intermediate data [10], [20]. Thus, the physical leakage becomes independent of the sensitive data. One of the most used countermeasure is Boolean masking [12]. Theoretically, when implemented correctly, it ensures a high level of security in that an attacker requires to capture & analyze a significant number of observations to recover the secret. However, in practice, and particularly for hardware designs, masked implementations are not as secure as expected. In fact, the formal model used to prove their resistance is not precise enough. Due to propagation time in logic gates, many transitions occur (a.k.a glitches) and [18], [19] pinpointed a first order leakage in the masked AND gate proposed in [15], [27]. To make masked (non-linear) gates secure, Threshold Implementation (TI) [13], [23] provides alternative schemes that resist against glitches. An efficient verification is then essential to check the security of the whole scheme. Recently, many tools that provide formal verification in presence of glitches have been proposed [2], [4], [6]. Those tools suppose a powerful attacker model as the one of d-probing model [15], and check stronger properties. Nevertheless, those properties may be stronger than necessary and incite designers to implement over-complex schemes.

*a) **Our Contributions.**:* Masking schemes ensure that sensitives variables are randomized. At a hardware level this property must hold true not only at each clock cycle, but also during the evaluation of the combinatorial logic. Due to the delay of signal propagation within combinatorial gates, intermediate values mixing previous and current states of signals may be computed. This phenomenon is termed "glitching", and has the negative property that those transitions depend on the sensitive (secret) value. State-of-the-art protections against glitches either attempt to remove them to the point no further leakage occurs through glitches, or to separate the combinatorial gates dealing with the masks and the masked data. Those two strategies ensure the absence of sensitive leakages through glitches by a (conservative) design methodology [21], [28].

Now, we claim that those methodologies are overkill. We formalize an algorithm to verify the absence of leakages despite glitches in arbitrary netlists. This algorithm checks that all possible glitch configurations are not leaking sensitive information. We leverage this algorithm to validate the security of masked netlists which are optimized (with respect to gate count) compared to state-of-the-art glitch-resistant masking schemes. We exhibit examples of netlists smaller than state-of-the-art ones. Those netlists do not follow the design principles of the state-of-the-art resistant logic styles. Tools have been proposed to check styles, and obviously, they report a leakage warning on our optimized design, but we show that those are *false positives*. Our methodology allows for an exact verification in such a way that it does not check for sufficient condition, *but it does check that each transition is properly "masked"*.

*b) **Outline.**:* The paper is organised as follows: in section II we recall the state-of-the-art concepts of masking, evaluation and we detail the attacker model. In section III, we give some definitions that will allow us to characterize the type of leakage and the possible ways to analyze and identify them. Then, in section IV, we present our approach through

some motivating examples. In section V we use the results of section IV to implement a more compact inversion over GF_{2^4} secure despite of the presence of glitches. Analyses on EM acquisitions are provided in section VI, where we show how the leakage gradually decreases until its total disappearance when the netlist is patched (and provably verified with our methodology).

II. Related work

A. Principle of masking

The main goal of a masking scheme is to randomize the manipulated data to make the physical leakage independent of the secret value. Many ideas have been suggested in the literature. It was first proposed in [10], [12] and further extensions have been also studied for different implementations. In [27], a masked AND gate at order one is designed using a Boolean sharing. An extendable version to any protection order was described in [15]. It is proven to be secure when the computations are performed in a sequential order, which can be sufficient for software implementations.

However, in hardware implementations, the order of evaluations is not always respected. Because of the propagation time of the logic gates, a first order SCA leakage can remain [18]. Thus the security of the masked gates of [15], [27] are compromised [11], [22]. To cure this vulnerability, the TI has been introduced in [23]. The main property of TI is the Non-Completeness one (TI Non-Completeness Property (TINC)). It forces all shares of the same variable not to be handled at the same time by the same function.

To reduce the number of shares necessary to achieve a first-order secure scheme, registers need to be inserted at the output of the non-linear functions. The Domain Oriented Masking (DOM) scheme comes with almost the same concept, but aims at reducing the number of needed registers and fresh random masks. Both approaches are secure against first order attacks. In [24] Reparaz et al. highlighted similarities between each scheme, and how to modify the classical multipliers ([15], [27]) to achieve secure versions like TI.

B. Robustness validation

To validate a first order masking countermeasure, it is necessary to check that each intermediate result is statistically independent of the secret data: this is the probing security model [15]. In a first step, a leakage can be simulated at the algorithmic level. This method is used to prove the effectiveness of the most known masking schemes presented previously. However, in hardware implementations, some leakage may remain because of physical defaults as shown in [18]. One of the most critical leakages is induced because of the propagation time inside logic gates. A logic gate evaluates the result for each new entry with a delay equal to the propagation time of the gate. When input signals of the gate do not change at the same time (within the propagation time of the gate), the output may change more than once. To model this phenomena, a simulation based analysis can be adopted to emulate the behaviour of each signal. This can be done on back-annotated (or Post Synthesis (PS)) netlists using a hardware simulator (at electrical or logical level) [1], [26]. It is notably not resilient to environmental changes.

Recently, in 2017 Bertoni et al. [4] introduced a methodology to evaluate the combinatorial part of a circuit in presence of glitches. Followed by a formal method [6] based on Fourier transform of the circuit gates, and a formal representation of the leakage. However, it is remarkably very slow and time consuming. A very similar way to analyse a circuit is presented in [2]. It uses a symbolic representation of the leakage, which is propagated through the combinatorial gates. It is known to be more efficient, and checks stronger properties of probing security [3]. Those tools actually instantiate an abstract version of the circuit, and build an image of the SCA leakage, but do not really consider a realistic timing information. By verifying the properties of d-probing model, that are extended for gadgets composition concept in [3], we can deduce whether or not the circuit is secure. In some cases, to reduce the complexity of the analysis, heuristics are used to over-estimate the leakage, and avoid false negatives (but this allows some false positives). It helps to locate any source of vulnerabilities, but does not explain how and why the leakage is exploitable.

III. Preliminaries

A. Notations

We denote by $(\oplus)$ and $(*)$ the XOR and AND operations on Boolean variables (lowercase) or vectors (uppercase) respectively. To indicate the inputs (A, M) and the output S of a gate, that implements a Boolean function f, we use functional notations $S = f(A, M)$. A delayed value of a signal is indicated by apostrophes (S'). When the intermediate value depends only on some (delayed) signals, they will be indicated on its arguments, (for example: $S' = f(A', M')$). In general, we use X for the secret data, M indicates the masks and A the masked data $A = X \oplus M$. We suppose also that the masks are uniformly distributed and cannot be guessed by an attacker. The expression of a gate output S can be expressed either with the tuple (A, M) or (X, M). To distinguish both, we index the latter with X. Thus we have:

$$S = f_X(X, M) = f(X \oplus M, M) = f(A, M).$$

The evaluation of a given design will use both notations (or expressions) to determine which variable is leaking, and where the vulnerability is located. Finally, we consider in our study the XOR and the AND gates as elementary ones. Any Boolean function can be seen as a multi-linear polynomial; hence it can be implemented using those two gates. Building secure masked AND is the most challenging task, since a masked XOR is only two XOR gates, that deal with the different shares independently.

B. Concepts

A formal based approach can be adopted to analyse the netlist by checking that all signals are independent of the secret data:

- For each gate output, deduce the corresponding Boolean expression f from the netlist;
- Use some criterion of independence to ensure that f_X is statistically independent of the secret variable X. This criterion can be deduced from a full formal representation like in [2], [6], or by an exhaustive evaluation of the conditional probabilities $\mathbb{P}(S|X)$. This probability (or distribution) must be the same whatever the value taken by X.

In terms of value, this is enough to ensure that each signal is independent of the secret. However, in terms of transitions this is not sufficient. The vulnerabilities introduced by glitches are directly related to the leakage introduced by transitions. In our context we consider two sources of exploitable vulnerabilities:

- Value based vulnerability: when a signal value is not independent of the secret value.
- Transition based vulnerability: when the activity (or transitions) of the signal depends on the secret.

To check the first vulnerability, the authors in [29] presented a relation between the Walsh Transform (WT) and the statistical dependency of a Boolean function with its variables. In the following, $A = (a_0, \cdots, a_{n-1}), M = (m_0, \cdots, m_{n-1}), X = (x_0, \cdots, x_{n-1})$ are binary vectors, with $A = X \oplus M$.

Definition 1 (WT (from [29])): Let f be a Boolean function:

$$X = (x_0, \ldots, x_{n-1}) \mapsto f(X),\ GF_2^n \mapsto GF_2.$$

The Walsh Transform $F = WT(f)$ of f is defined as:

$$GF_2^n \mapsto \mathbb{Z},\quad F : W \mapsto \sum_{X \in GF_2^n} f(X)(-1)^{W \cdot X}$$

where

$$W \cdot X = \bigoplus_{i=0}^{n-1} w_i * x_i$$

is the standard scalar product.

Theorem 1 (Correlation immunity [29]): The Boolean combining function f for n binary variables is m^{th}-order correlation immune, where $1 \leq m \leq n\ iff$ its Walsh transform F satisfies:

$$\forall\ W \in GF_2^n,\ 1 \leq HW(W) \leq m,\ F(W) = 0.$$

Where HW denotes the Hamming Weight (HW).

Corollary 1: A function f_X is statistically independent of X, if it is independent of each subset of the involved secret variables. In general, if f_X is expressed as:

$$f_X(x_0, \ldots, x_{n-1}, m_0, \ldots, m_{n-1})$$

then, f_X is statistically independent of X iff:

$$\forall\ W\ \in\ GF_2^{2n},\quad F_X(W) = 0,$$
$$\text{where } w_n = \ldots = w_{2n-1} = 0.$$

Definition 2 (Security with respect to value): A Boolean function $f(A, M)$ is secure in terms of value if it is statistically independent of $X = A \oplus M$ (i.e it satisfies corollary 1).

This gives a spectral equivalent version to check if any Boolean function is statistically dependent on any set of secret variables. As an example, if for $W = (1, 1, 0, \cdots, 0)$ we have $F(W) \neq 0$, then f depends on (x_0, x_1). Nevertheless, theorem 1 cannot be used directly to check if a given function is secure in terms of transitions. To take transitions into account, we need to consider two successive states of the signal.

Definition 3 (Transition leakage): We define the transition leakage as the distance between two successive values of a function f by $D_{\delta_A,\delta_M}(f, A, M)$ for some δ_A, $\delta_M \in GF_2^n$:

$$D_{\delta_A,\delta_M}(f, A, M) = f(A \oplus \delta_A, M \oplus \delta_M) \oplus f(A, M)$$

This distance is characterized by $\delta = (\delta_A, \delta_M)$, which is a bit-vector such that the bits at one indicate the delayed signals. The delayed variables are then, $A' = A \oplus \delta_A$ and $M' = M \oplus \delta_M$.

IV. Formalization of netlist static analysis

In the following, we give some examples to introduce our security verification methodology. Mainly we apply the notions described previously to analyse non-linear functions in presence of glitches. In section IV-A, we analyse the impact of a glitch at the netlist inputs, and in section IV-B, we extend this approach to netlist logic, and give a complete formal model that proves security in presence of glitches. Finally, in section IV-D, we give a simple masked AND gate that achieves our security criteria.

A. Motivating examples

Example 1 (Vulnerable design): Let $X = (x_0, x_1, x_2)$, $M = (m_0, m_1, m_2)$ and $A = X \oplus M$, and f defined as:

$$f(A, M) = (m_2 \oplus a_0 * m_1) \oplus a_1 * m_0$$

which implements an equivalent component of the masked AND gate described in [27]. We can easily check that f is uniformly distributed in terms of value and independent of X (i.e $\mathbb{P}(f = 1|X) = \frac{1}{2}$). However, in the case of a transition when $\delta_A = (0, 1, 0)$ and $\delta_M = (0, 1, 0)$, we get:

$$D_{\delta_A,\delta_M}(f, A, M) = a_0 \oplus m_0 = x_0$$

which depends on X. Thus, this implementation is vulnerable in terms of transition, and may leak in presence of glitches. Besides, it leaks x_0 only if the timing characteristic of the device will have the couple (a_1, m_1) arrive after the other signals, and the transitions $(a_1, m_1) \rightarrow (a_1', m_1')$ are seen (almost) at the same time from the last XOR gate (red color). Another interesting case is when $\delta_A = (1, 1, 0)$ and $\delta_M = (1, 1, 0)$:

$$D_{\delta_A,\delta_M}(f, A, M) = x_0 \oplus x_1.$$

In this case, the leakage is not correlated to the HW of X. This is in fact the general form of the leakage created by glitches when the multi-linear polynomial of the gate is of degree 2. In this configuration, it results in the WT of f that $F_X(W) \neq 0$ for $W = (1, 1, 0, \cdots, 0)$. The same holds actually for the traces of power consumption (see section VI).

Example 2 (Secure design): Here we consider another case that involves also both shares of the same variable. We have: $X = (x_0, x_1, x_2, x_3)$, $M = (m_0, m_1, m_2, m_3)$, $A = X \oplus M$, and:

$$f(A, M) = a_0 * (m_1 \oplus m_2) \oplus a_1 * (m_0 \oplus m_3).$$

By analyzing all the possible transitions $\forall\ \delta_A, \delta_M \in GF_2^4$, we have always f_X independent of X. We can see that if both (m_1, m_2) (or (m_0, m_3)) change, f do not change, so the number of transitions to compute can be reduced. The relevant value of f' are:

$$f(A \oplus \delta_A, M \oplus \delta_M) = (a_0 \oplus \delta_0) * (m_1 \oplus m_2 \oplus \delta_1) \oplus (a_1 \oplus \delta_2) * (m_0 \oplus m_3 \oplus \delta_3)$$

with $\delta_i \in GF_2$, and $\delta_0 = \delta_{a_0}$, $\delta_2 = \delta_{a_1}$, $\delta_1 = \delta_{m_1} \oplus \delta_{m_2}$, $\delta_3 = \delta_{m_0} \oplus \delta_{m_3}$.

If $\delta_A = (0, 1, 0, 0)$ and $\delta_M = (0, 1, 0, 0)$ then:

$$D_{\delta_A, \delta_M}(f, A, M) = x_0 \oplus m_3$$

which is uniform and independent of X. We can deduce that this is secure even in presence of glitches, but according to [2], [6] this function is not secure, because f uses all shares of the secret variables $x_0 = (a_0 \oplus m_0)$ and $x_1 = (a_1 \oplus m_1)$.

B. Formal model - Glitch-extension

In some cases, glitches can be induced because of the same input signal due to an internal delay. If S has multiple paths to the same gate, it may induce multiple transitions at the final result. To take into account this behaviour, we introduce the notion of transient input.

Definition 4 (Transient input): The transient input $(\tilde{A}, \tilde{M})$ of a function f is the set of all variables that come from different combinatorial paths. Thus, the same variable at different place are considered independently. We denote also by $\tilde{f}$ the transient expression of f. Thus, we have: $\tilde{f}(\tilde{A}, \tilde{M}) = f(A, M)$.

To understand clearly this notion, it is more appropriate to use expression tree, namely the prefix traversal.

Definition 5 (Prefix traversal): A prefix representation is the expression produced when placing the operator first and the two operands next.

In example 2, f can be expressed as:

$$f(A, M) = \oplus(*(a_0, \oplus(m_1, m_2)), *(a_1, \oplus(m_0, m_3))) \quad (1)$$

The advantage of this notation lies in the fact that it conserves the structure of how the function is instantiated in the real circuit. For instance, eq. (1) can be instantiated equivalently by:

$$f(A, M) = \oplus(\oplus(*(a_0, m_1), *(a_0, m_2)), \oplus(*(a_1, m_0), *(a_1, m_3)))$$

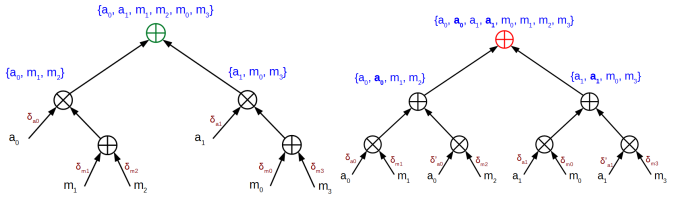
which gives different leakage results when considering glitches. More precisely, each leaf of the tree is considered independently in the case of transitions, thus different δ could be associated to the same variable.

Proposition 1 (Security with respect to transition (Glitch-Extended Security)): Let f be the expression of the signal S, and $\tilde{A}, \tilde{M} \in GF_2^{\tilde{n}}$ the transient input variables of S. S is secure against glitches iff for any $\delta_a, \delta_m \in GF_2^{\tilde{n}}$:

$$D_{\delta_a, \delta_m}(\tilde{f})(\tilde{A}, \tilde{M})$$

is statistically independent of $X = A \oplus M$.

Proof: In fact, $D_{\delta_a, \delta_m}(\tilde{f})(\tilde{A}, \tilde{M})$ is a Boolean function therefore, theorem 1 and corollary 1 apply directly. ■



(a) (b)

Fig. 1: Different ways to implement f of example 2. Transient inputs for each gate are shown in blue: in (a) $(\tilde{A}, \tilde{M}) = ((a_0, a_1), (m_0, m_1, m_2, m_3))$, (b) $(\tilde{A}, \tilde{M}) = ((a_0, a_0, a_1, a_1), (m_0, m_1, m_2, m_3))$. For each input we associate different delay (δ).

In fig. 1, we show the two different ways to implement f introduced in example 2. In the first case (section IV-B), f is not leaking X. In the second case ((section IV-B)), f may leak X. The reason is that, the variable a_0 (resp. a_1) may impact the function f differently (at two different time samples).

C. Our leakage detection algorithm

Algorithm 1 scans the netlist (input S) and first checks whether each node is masked, and then tests whether it is vulnerable to glitches. If a configuration yields an unbalanced distribution, then the algorithm returns the corresponding δ and the leaking signal. The internal functions work as follows:

- $get\_transient\_inputs$: returns the inputs of each gate as instantiated in the design (definition 4).
- $get\_masked\_variables$: returns the masked variables of the input gate.
- $get\_masks\_variables$: returns the masks variables of the input gate.

If all transitions do not depend on the sensitive variable X, then algorithm 1 returns "Secure".

We insist that this verification methodology is agnostic in the actual quantitative delays within the netlist, because we abstract away the glitching source as an anticipated evaluation anywhere in the netlist. Our threat model is that the netlist is known, represented as a tree of gates, and is immutable. The attacker can well probe a node, but cannot alter the netlist by cutting wires or disabling gates.

We show in table I a comparison of our approach with other existing formal analysis method.

In the following sub-section, we build a masked AND gate secured against glitches, based on the previous observation

Algorithm 1: Security Verification Against Glitches.

Input: S: the design, A: List of masked variables, M: List of mask variables

Output: "Secure" or first leaking signal

for $s \in S$ **do** // For each signal s in the netlist
  $transient\_inputs\_of\_s \leftarrow get\_transient\_inputs(s)$
  $n \leftarrow \|transient\_inputs\_of\_s\|$
  $M_s \leftarrow get\_masks\_variables(transient\_input\_of\_s, M)$
  $A_s \leftarrow get\_masked\_variables(transient\_input\_of\_s, A)$
  $X \leftarrow A_s \oplus M_s$
  $f \leftarrow s(inputs\_of\_s)$
  $f_X \leftarrow f(X \oplus M_s, M_s)$
  $value\_distribution \leftarrow \mathbb{P}(f_X|X)$ // Security in terms of value
  if *value_distribution is not balanced* **then**
    return s // First order leaking signal s in terms of value
  for $\delta \in GF_2^n$ **do** // Security in terms of transition
    $f' \leftarrow s(transient\_inputs\_of\_s \oplus \delta)$
    $T \leftarrow f \oplus f'$ // T is the transition
    $T_X \leftarrow T(X \oplus M_s, M_s)$
    $distribution \leftarrow \mathbb{P}(T_X|X)$
    if *distribution is not balanced* **then**
      return s, δ // s being the leaking signal, and δ indicating the delayed signal
return "Secure"

TABLE I: Comparison with state-of-the-art formal analysis methods

Analysis method	Leakage location	Value leakage model	Exact transient leakage	Formal leakage expression
[2]	✓	✓	✗	✗
[6]	✓	✓	✗	✗
This paper	✓	✓	✓	✓

made in section IV-B. This version does not have a major advantage over TI, but we use the same principle for more concrete cases where the difference is more significant in terms of area (section V).

D. Our glitch-resistant masked AND gate

Let $a_i = x_i \oplus m_i$ and $b_i = y_i \oplus n_i$ for $i \in \{0, 1\}$. In table II, we give the different steps for implementing the masked AND gate. The left one is insecure. The right one satisfies our security model against glitches, and also in terms of value. To check

TABLE II: Masked implementation of AND gate. z_i are fresh random. Left: we have $x_0 * y_0 = i_4 \oplus z_0$; Right: we have $x_0 * y_0 = T_1 \oplus T_2 \oplus T_3$.

Vulnerable masked AND [27]	Our secure masked AND
$s_1 \leftarrow a_0 * b_0$	$s_1 \leftarrow (n_0 \oplus z_0)$
$s_2 \leftarrow a_0 * n_0$	$s_2 \leftarrow (m_0 \oplus z_1)$
$s_3 \leftarrow b_0 * m_0$	$s_3 \leftarrow a_0 * s_1$
$s_4 \leftarrow m_0 * n_0$	$s_4 \leftarrow b_0 * s_2$
$i_1 \leftarrow z \oplus s_1$	$i_1 \leftarrow b_0 \oplus z_0$
$i_2 \leftarrow i_1 \oplus s_2$	$i_2 \leftarrow a * i_1$
$i_3 \leftarrow i_2 \oplus s_3$	$i_3 \leftarrow b * z_1$
$i_4 \leftarrow i_3 \oplus s_4$	$T_1 \leftarrow s_2 \oplus s_4$
	$T_2 \leftarrow i_2 \oplus i_3$
	$T_3 \leftarrow m_0 * n_0$

that, let's consider a non-linear function f, defined as:

$$f(A, B, M, N, Z) = \oplus(*(a_0, \oplus(n_0, z_0)), *(b_0, \oplus(m_0, z_1))). \tag{2}$$

We can see that both shares of the secret (x_0, y_0) are manipulated by f. We have seen in section IV-B that f is secure according to algorithm 1. Particularly, as a case of comparison with the classical masked AND gate when:

$$(a_0, b_0, m_0, n_0) \rightarrow (a_0 \oplus 1, b_0 \oplus 1, m_0 \oplus 1, n_0 \oplus 1)$$

we have:

$$(i_1') \rightarrow (i_1) \equiv x_0 \oplus y_0$$

and for f in eq. (2) we get (with $f_0 = a_0 * (n_0 \oplus z_0)$ and $f_1 = b_0 * (m_0 \oplus z_1)$):

$$(f_0' \oplus f_1') \rightarrow (f_0 \oplus f_1) \equiv x_0 \oplus y_0 \oplus m_1 \oplus n_1,$$

which is not vulnerable. Whatever the considered transition, either the result depends only on one share of (x_0, y_0), or it is masked by n_1 or m_1. In other words, each transition is masked at least with one mask n_i or m_i.

We can implement a masked AND gate (without any resharing of the inputs) using two fresh random z_0 and z_1 (we can reuse masks of other variables to reduce the usage of randomness):

$$\begin{aligned} T_1 &= a_0 * (n_0 \oplus z_0) \oplus b_0 * (m_0 \oplus z_1), \\ T_2 &= a_0 * (b_0 \oplus z_0) \oplus b * z_1, \\ T_3 &= m_0 * n_0 \end{aligned} \tag{3}$$

Thus, the output result is $x_0 * y_0 = T_1 \oplus T_2 \oplus T_3$. Incidentally, we can see that T_1 satisfies proposition 1 (cf. example 2), T_2 and T_3 satisfy the TINC property.

V. Practical Case: Masked Inversion in GF_{2^4}

We design now a complete implementation of a GF_{2^4} inverter, based on the first-order masked Canright version of the Advanced Encryption Standard (AES) Substitution Box (S-Box). In section VI, we give the full implementation of our S-Box, integrating our GF_{2^4} inverter. In the same section, we compare our formal results with the results of digital simulations at Register Transfer Level (RTL) and PS level.

For the sake of clarity, we focus our analyses on the GF_{2^4} inverter. The results can be transposed to the operations performed in GF_{2^8} inverter.

We detail the formal expression of each signal and explain how the leakage is created. Subsequently, we propose a possible fix, and constantly check the security of the design until no leakage is reported.

A. Canright AES S-Box

Canright proposed an optimized instance of the AES S-Box [9] based on standard CMOS gates XOR, NOR and $NAND$. The inversion is computed over the Tower Field representation of GF_{2^8}. The inversion of an element in GF_{2^8} can be reduced to one inversion in GF_{2^4}, some multiplications and additions in GF_{2^4} and GF_{2^2}. This implementation takes the masked input, the input mask, and the output mask, 8 bits each. We can find symmetry in the operations performed over GF_{2^4} inside the GF_{2^8} inverter, and those performed over GF_{2^2} inside the GF_{2^4} inverter. Thus, the GF_{2^4} inverter takes, 3 inputs of 4 bits (masked input, input mask and output mask).

B. Formal based evaluation of Canright inverter

If we explicitly write the expression of the inputs of csa gate, we get (for one bit, namely bit number 1):

$$\begin{aligned}
an_1 &= (a_1 * n_1) \oplus ((a_0 \oplus a_1) * (n_0 \oplus n_1)) \\
mb_1 &= (m_1 * b_1) \oplus ((m_0 \oplus m_1) * (b_0 \oplus b_1)) \\
cst_1 &= a_1 \oplus b_1 \oplus a_1 * b_1 \oplus (a_1 \oplus a_0) * (b_1 \oplus b_0) \oplus N_3 \\
csa_1 &= cst_1 \oplus an_1, \\
csb_1 &= csa_1 \oplus mb_1
\end{aligned}$$

where N_3 is a fresh mask (one bit of the output mask). These equations are also represented as a netlist in fig. 2.

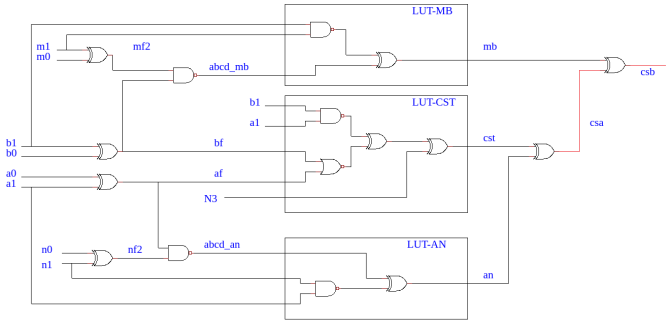


Fig. 2: Masked circuit computing csb_1. The leaking signals (red color) are csa_1 and csb_1.

The order of summation is also important, if an_1 and mb_1 are summed together, the result will depend on X:

$$\begin{aligned}
an_1 \oplus mb_1 = &(x_2 * n_0 \oplus x_2 * n_1 \oplus x_3 * n_0) \\
&\oplus (x_0 * m_0 \oplus x_0 * m_1 \oplus x_1 * m_0) = S_{ab}.
\end{aligned}$$

Obviously, $\mathbb{P}(S_{ab}|X) \neq \mathbb{P}(S_{ab})$, particularly for $X = 0$, we have $S_{ab} = 0$ with probability 1. Now, let us consider the case where all signals are summed in the right order. For example, the signal csa ($csa = cst \oplus an$). For the first bit, we have:

$$\begin{aligned}
csa_1 = &a_0 * (b_0 \oplus b_1) \oplus a_1 * b_0 \oplus a_1 \\
&\oplus b_1 \oplus a_1 * n_0 \oplus a_0 * (n_0 \oplus n_1)) \oplus N_3.
\end{aligned}$$

In terms of value, the result is protected (at least) by the fresh mask N_3. However, in terms of transition in presence of propagation time, a_0 can arrive with some delay and the transition ($csa'_1 \rightarrow csa_1$) will leak ($x_0 \oplus x_1$). According to algorithm 1 when $\delta_{a_0} = 1$:

$$csa'_1 \oplus csa_1 = (a'_0 \oplus a_0) * (b_0 \oplus b_1 \oplus n_0 \oplus n_1) = x_0 \oplus x_1.$$

This depends on X, hence **the Canright design is not secure**. Note however that, this leakage model is not conventional. Only a thorough analysis and dedicated attacks can exploit this kind of leakage, such as collision or template. Actual exploitation of this first-order flaw is detailed in section VI-A.

C. Our GF_{2^4} inverter, compact and provably secure

In the following, we show how the inversion can be achieved within only one cycle. Then, using the observation of eq. (3), we reduce the number of needed registers (FF).

1) Inversion in GF_{2^4}: First we express the equations of the inverse $y = (y_0, \ldots, y_3)$ of any element $x = (x_0, \ldots, x_3) \in GF_2^4 \simeq GF_{2^4}$:

$$\begin{aligned}
y_0 &= x_1 * x_2 * x_3 \oplus x_0 * x_2 \oplus x_0 * x_3 \oplus x_1 * x_3 \oplus x_2 \\
y_1 &= x_0 * x_2 * x_3 \oplus x_0 * x_3 \oplus x_1 * x_3 \oplus x_2 \oplus x_3 \\
y_2 &= x_0 * x_1 * x_3 \oplus x_0 * x_2 \oplus x_1 * x_2 \oplus x_1 * x_3 \oplus x_0 \\
y_3 &= x_0 * x_1 * x_2 \oplus x_1 * x_2 \oplus x_1 * x_3 \oplus x_1 \oplus x_0
\end{aligned}$$

The masked result can be deduced by replacing x_i by $a_i \oplus m_i$. For the first bit y_0 we get:

$$y_0 = S_1 \oplus S_2 \oplus S_3 \oplus S_4 \oplus S_5 \oplus S_6 \oplus S_7 \oplus S_8 \qquad (4)$$

with,

$$\begin{aligned}
S_1 &= a_2 * a_3 * a_1 \oplus a_2 * a_0 \oplus a_3 * a_1, \\
S_2 &= a_2 * a_3 * m_1 \oplus a_2 * m_0 \oplus a_3 * m_0, \\
S_3 &= a_2 * a_1 * m_3 \oplus a_0 * m_3 \oplus a_2, \\
S_4 &= a_2 * m_3 * m_1 \oplus a_3 * m_1, \\
S_5 &= a_3 * a_1 * m_2 \oplus a_0 * m_2, \\
S_6 &= a_3 * m_2 * m_1 \oplus a_3 * a_0 \oplus m_2, \\
S_7 &= a_1 * m_2 * m_3 \oplus a_1 * m_3 \oplus m_3 * m_0, \\
S_8 &= m_2 * m_3 * m_1 \oplus m_2 * m_0 \oplus m_3 * m_1
\end{aligned}$$

We can see that each result S_i respects the TINC. Moreover, as each monomial of degree 3 cannot be combined with any other monomial of degree 3, the minimal number of shares that respect TINC is 8. To achieve the inversion in one cycle, 8 FFs and 8 fresh random are needed to remask each S_i. We note that each y_i can be expressed in the same way as eq. (4).

2) Reducing the number of registers: To reduce the number of needed FFs, we need to optimize the masked computation of monomials of degree 3. For y_0 we have:

$$x_1 * x_2 * x_3 = (a_1 \oplus m_1) * x_2 * x_3$$
$$\begin{aligned} a_1 * x_2 * x_3 &= a_1 * (a_2 * a_3 \oplus a_2 * m_3 \oplus a_3 * m_2 \oplus m_2 * m_3) \\ &= a_1 * (a_2 * (m_3 \oplus z_0) \oplus a_3 * (m_2 \oplus z_1)) \\ &\oplus a_1 * (a_2 * (a_3 \oplus z_0) \oplus a_3 * z_1) \oplus a_1 * m_2 * m_3. \end{aligned}$$

The same thing holds for m_1. Thus, we reduce the number of needed FFs to 6. Finally, the masked computation of the LSB of the inverse in GF_{16} is implemented as:

$$y_0 = S_1 \oplus S_2 \oplus S_3 \oplus S_4 \oplus S_5 \oplus S_6 \tag{5}$$

with,

$$S_1 = a_1 * (a_2 * (m_3 \oplus z_0) \oplus a_3 * (m_2 \oplus z_1))$$
$$S_2 = m_1 * (a_2 * (m_3 \oplus z_0) \oplus a_3 * (m_2 \oplus z_1))$$
$$S_3 = a_1 * (a_2 * (a_3 \oplus z_0) \oplus a_3 * z_1 \oplus a_3) \oplus a_0 * (a_2 \oplus a_3) \oplus a_2$$
$$S_4 = m_1 * (a_2 * (a_3 \oplus z_0) \oplus a_3 * z_1 \oplus a_3) \oplus m_0 * (a_2 \oplus a_3)$$
$$S_5 = a_1 * (m_2 * m_3 \oplus m_3) \oplus a_0 * (m_2 \oplus m_3)$$
$$S_6 = m_1 * (m_2 * m_3 \oplus m_3) \oplus m_0 * (m_2 \oplus m_3) \oplus m_2$$

Note that each signal S_i satisfies corollary 1 and proposition 1 and hence, algorithm 1 returns "Secure" for each S_i. To ensure a secure compression, each S_i needs to be remasked with a fresh mask and stored into a register ($S_{i_{ff}} \leftarrow S_i \oplus z_j$). At most, 8 new fresh masks are needed. For each bit y_i, the positions of the masks z_j can be changed such that the output mask of each bit would be different. The number of possible output masks is: $\binom{8}{6} = 28$.

We synthesized the GF_{2^4} inversion, using the Cadence GSCLIB045 standard cell demonstration library, without any timing constraint. The comparison metric is the Gate Equivalent (GE) relative to the NAND2X1 cell of the library.

TABLE III: GF_{2^4} inverter - Comparing areas (GE)

Implementation	GE (logic)	GE (sequential)	#Cycles	First-order security	
				Value	Glitch
Canright [8]	153	0	0	✓	✗
DOM [13]	358	144	2	✓	✓
TI [5]	618	/	1	✓	✓
This paper (eq. (5))	296	127	1	✓	✓

The reference design is a part of the simple Canright design from [8]. As shown in table III, the combinational area roughly double, and 30% of more area is added for the registers. Compared with the DOM version (without pipeline), our version is 19% smaller. The TI implementation from [5] takes much more area. The number of GE is taken from the publication and not issued from the same library, but it still huge compared with DOM and our version.

In fig. 3 we show the architecture of a one bit inversion. Each S_i is remasked with a new fresh mask before registration

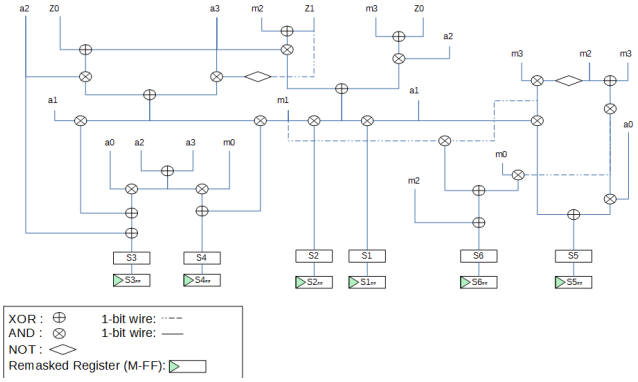


Fig. 3: Our new design of one bit GF_{2^4} inversion - Formally proven to resist against glitches.

(green registers M-FF). We have proven by netlist traversal algorithm (algorithm 1) that each signal in the design verifies corollary 1 for security in terms of value, and proposition 1 for security in terms of transitions (glitches).

VI. Actual Exploitation of Vulnerable Netlists

In this section, we demonstrate attacks on netlists which have been demonstrated to contain vulnerabilities. First, in section VI-A, we show some attacks on simulated traces. We use Correlation Power Analysis (CPA) and Collision-Correlation Power Analysis (CCA) for the exploitation phase. Second, in section VI-C and section VI-D, we demonstrate some vulnerabilities on measured EM traces, using the Normalized Inter-Class Variance (NICV) metric.

A. SCA evaluation of Canright inverter - Digital simulation

Firstly, we have analysed the Canright RTL code based on digital simulation. We have confirmed that all intermediate results are correctly masked and independent of the secret data. Secondly, the same analysis was performed on a synthesized netlist using a SAKURA-G Field Programmable Gate Arrays (FPGA) target (without timing), and no leakage has been reported. Once again, all combinatorial signals are independent of the secret data, and the synthesizer did not make any optimisation that may unmask the secret data. This is consistent with our constraints: we have forced the synthesizer to keep all intermediate signals and the hierarchy of each module.

Finally, when we added the timing information to the netlist, the tool has reported several leaking signals. For instance, the first leaking level was at the compression step of the multipliers outputs, similarly to the case of the classical multiplier.

The first reported leakage in the design was the signal csa (see fig. 2). This signal is the result of a XOR of the output of two non-linear functions that deal with some identical shared data.

Based on the simulation results, we were able to explicitly specify the timing information on the SDF file. For this case, we removed all the timing information except those of a_0. Each signal is sampled at 10 ps, which allows to capture all the transitions independently.

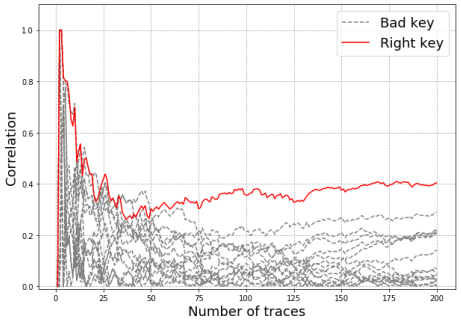


Fig. 4: CPA on csa_1 activity. Only 75 traces are sufficient to recover the secret key.

As expected, the leakage was correlated to $(x_0 \oplus x_1)$. In fig. 4, we show the result of the CPA using the leakage model returning $(x_0 \oplus x_1)$, the red curve shows the result of the right key. We get the same results using CCA. The leakage model $\mathcal{L}$ is computed for any key hypothesis K and the (known) output $C \in GF_2^4$ as the following:

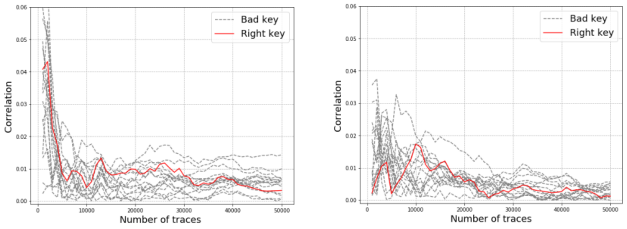
$$\begin{aligned} X &= (c \oplus K)^{-1} \in GF_2^4 \\ \mathcal{L}(C, K) &= x_0 \oplus x_1. \end{aligned} \tag{6}$$

We recall that, by convention, the inverse of 0 is mapped to 0 itself.

As explained in section II, this leakage is created only if the change induced by a_0 would influence the gate csa at the same time (within the propagation time of the XOR LUT).

B. SCA evaluation of our inverter - Digital simulation

The first and the second order CPA based on simulated traces of our design (presented in section V-C) is shown in fig. 5.



(a) (b)

Fig. 5: First (a) and second (b) order CPA on S_1 of eq. (5). The right key correlation is not distinguishable.

The result of the first-order CPA confirms that the secret key is indistinguishable (see fig. 5a). Besides, even the second-order CPA is not effective for instance (fig. 5b). The reason is that, there is no configuration (a couple (δ_a, δ_m)) where a given mask $\{m_i\}$ can leak alone (in terms of transition). If the expression of the leakage involves one mask m_i, it also involves one mask z_i. Thus, the combination of the leakage cannot depend on x_i because of the extra fresh mask z_i.

C. EM analysis of GF_{2^4} inverter

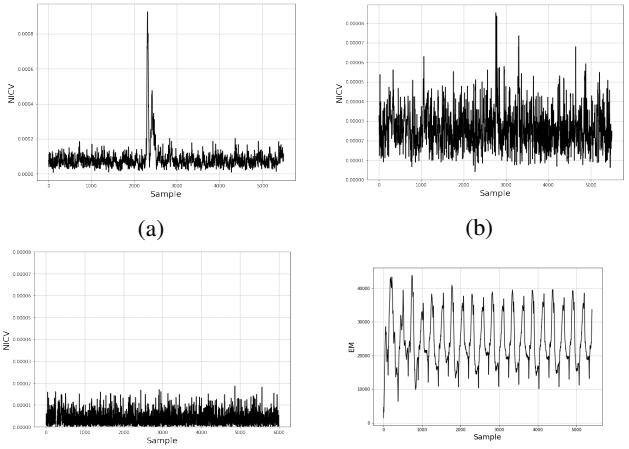
To perform a real evaluation with a best characterisation of the leakage, we have implemented a small substitution function $Sbox^*$ using two GF_{16} inverters, thus we get a small block encryption that we note AES^*, with:

$$Sbox^*(x_7, \cdots, x_0) = ((x_7, \cdots, x_3)^{-1}, (x_3, \cdots, x_0)^{-1})$$

In the following, we give the results of our leakage characterisation on EM measurements. For each implementation, we have different versions of the GF_{16} inverter; with and without registers as presented in section V-C:

- AES_0^*: no register in the GF_{16} inverter.
- AES_1^*: only 3 registers are used instead of 6, we register only $S_r + S_{r+1}$ for $r \in \{0, 2, 4\}$.
- AES_2^*: implementation of section V-C (fig. 6c). Please, refer to eq. (5) for more understanding.

The different versions have the same number of cycles per round.



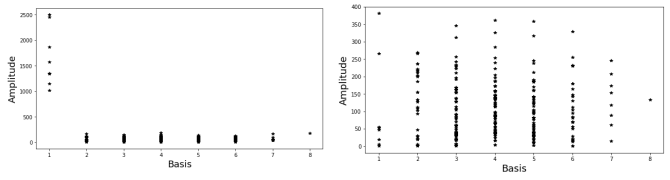
(a) (b) (c) (d)

Fig. 6: NICV based on an unmasked intermediate state of AES^*: (a) AES_0^* with 200 000 traces, (b) AES_1^* with 600 000 traces, (c) AES_2^* with 1 000 000 traces and (d) a raw trace of AES_1^* (first three rounds).

In fig. 6, we show how the leakage is progressively removed by inserting registers (fig. 6c) for the different implementations of AES^*. Comparing the results of fig. 6a and fig. 6b, the leakage of AES_1^* is 10 times smaller than AES_0^*, whereas AES_2^* does not show any visible leakage.

To mark the difference between a leakage easily characterized by a HW leakage model, we applied the WT transform to the EM traces [14].

The results are displayed in fig. 7. For a leakage that can be exploited by a HW model, the amplitude is more significant when the base is equal to $HW(w_0, \cdots, w_n) = 1$ (fig. 7a). On the other hand, when the leakage is due to glitches (case of fig. 6a), all the bases are almost equivalent. So the leakage is a mixture of bits, like the one identified in section IV.



(a) (b)

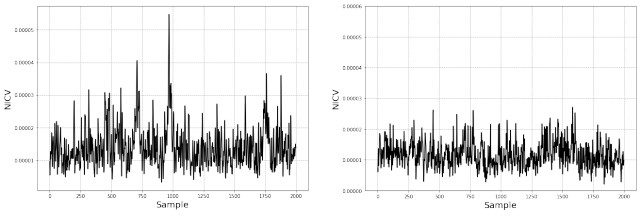
Fig. 7: WT applied to EM traces: (a) HW leakage model case, (b) Glitches leakage case. $Basis = HW(W)$.

D. EM analysis of AES design: S-Box

For the full AES implementation, we have added registers at the output of each GF_{16} multiplier (see figure fig. 10). As previously, to observe the evolution of the leakage, we implemented and analysed two versions of AES:

- AES_1: only 3 registers are used instead of 6, like AES_1^*.
- AES_2: implementation of section V-C.

They are equivalent to AES_1^* and AES_2^*, described in the previous section.



(a) (b)

Fig. 8: NICV based on an unmasked intermediate state of AES: (a) AES_1 with 1 000 000 traces – very low leakage (comparable to fig. 6b), (b) AES_2 wiht 1 200 000 traces – no leakage is visible.)

In fig. 8 we show the results of the NICV for our different implementations of AES. In fig. 8a, we can note that the leakage that we have identified in fig. 6b is more difficult to detect because of the extra activity (noise) of the rest of the AES S-Box, where fig. 8b do not show any leakage like for fig. 6c.

VII. Conclusion

In this paper, we have evaluated the security of hardware masked implementations against SCA vulnerabilities in presence of glitches. We have detailed the form of the leakage and exposed the different ways to prevent information leakage.

Namely, we present an algorithm to check exactly for leakage in terms of values and transitions in masked netlists. It is subsequently possible to design more compact and optimized functions. Indeed, our algorithm allows to check the security of netlists implementing logic using gadgets which are less constrained than the conservative methodology required by TI or DOM.

We have given more understanding about the leakage of masked non-linear gates based on in-depth analyses in terms of transition based power consumption. Thus, we have identified the critical parts on the non-linear gates that should be treated carefully. In addition to a formal security proof, our results are argued through empirical verification on simulated synthesised netlist, as it was expected from the formal analysis in terms of transition in presence of propagation time.

As a perspective, it is natural to extend this approach to higher-order masking, while comparing the advantages and disadvantages with more restrictive and abstract models. It would also be necessary to model other physical phenomena, such as coupling and technological dispersion that can induce further exploitable leakages with, potentially, a very large number of traces.

Acknowledgments

This work has been partly financed via the BRAINE Project from European Union's Horizon2020 / ECSEL research and innovation program, under grant agreement N° 876967. The results have been integrated in the Catalyzr tool [25].

References

[1] Alam, M., Ghosh, S., Mohan, M., Mukhopadhyay, D., Chowdhury, D.R., Gupta, I.: Effect of glitches against masked AES S-box implementation and countermeasure. IET Information Security **3**(1), 34–44 (2009)

[2] Barthe, G., Belaïd, S., Cassiers, G., Fouque, P.A., Grégoire, B., Standaert, F.X.: maskverif: Automated verification of higher-order masking in presence of physical defaults. In: European Symposium on Research in Computer Security. pp. 300–318. Springer (2019)

[3] Barthe, G., Belaïd, S., Dupressoir, F., Fouque, P.A., Grégoire, B., Strub, P.Y., Zucchini, R.: Strong non-interference and type-directed higher-order masking. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 116–129 (2016)

[4] Bertoni, G., Martinoli, M., Molteni, M.C.: A methodology for the characterisation of leakages in combinatorial logic. Journal of Hardware and Systems Security **1**(3), 269–281 (2017)

[5] Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., Rijmen, V.: A more efficient AES threshold implementation. In: International Conference on Cryptology in Africa. pp. 267–284. Springer (2014)

[6] Bloem, R., Groß, H., Iusupov, R., Könighofer, B., Mangard, S., Winter, J.: Formal verification of masked hardware implementations in the presence of glitches. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 321–353. Springer (2018)

[7] Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: International Workshop on Cryptographic Hardware and Embedded Systems. pp. 16–29. Springer (2004)

[8] Canright, D.: David Canright's tiny AES S-boxes, Verilog structural code of the netlist: https://github.com/coruus/canright-aes-sboxes

[9] Canright, D., Batina, L.: A Very Compact "Perfectly Masked" S-Box for AES. In: ACNS. Lecture Notes in Computer Science, vol. 5037, pp. 446–459 (2008)

[10] Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Annual International Cryptology Conference. pp. 398–412. Springer (1999)

[11] Clavier, C., Feix, B., Gagnerot, G., Roussellet, M., Verneuil, V.: Improved collision-correlation power analysis on first order protected AES. In: International Workshop on Cryptographic Hardware and Embedded Systems. pp. 49–62. Springer (2011)

[12] Coron, J.S., Goubin, L.: On boolean and arithmetic masking against differential power analysis. In: International Workshop on Cryptographic Hardware and Embedded Systems. pp. 231–237. Springer (2000)

[13] Groß, H., Mangard, S., Korak, T.: Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. IACR Cryptology ePrint Archive **2016**, 486 (2016)

[14] Guilley, S., Heuser, A., Ming, T., Rioul, O.: Stochastic side-channel leakage analysis via orthonormal decomposition. In: International Conference for Information Technology and Communications. pp. 12–27. Springer (2017)

[15] Ishai, Y., Sahai, A., Wagner, D.: Private circuits: Securing hardware against probing attacks. In: Annual International Cryptology Conference. pp. 463–481. Springer (2003)
[16] Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Annual international cryptology conference. pp. 388–397. Springer (1999)
[17] Kocher, P.C.: Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In: Annual International Cryptology Conference. pp. 104–113. Springer (1996)
[18] Mangard, S., Popp, T., Gammel, B.M.: Side-channel leakage of masked CMOS gates. In: Cryptographers' Track at the RSA Conference. pp. 351–365. Springer (2005)
[19] Mangard, S., Schramm, K.: Pinpointing the side-channel leakage of masked AES hardware implementations. In: International Workshop on Cryptographic Hardware and Embedded Systems. pp. 76–90. Springer (2006)
[20] Messerges, T.S.: Securing the aes finalists against power analysis attacks. In: International Workshop on Fast Software Encryption. pp. 150–164. Springer (2000)
[21] Moradi, A., Mischke, O.: Glitch-free implementation of masking in modern fpgas. In: 2012 IEEE International Symposium on Hardware-Oriented Security and Trust. pp. 89–95. IEEE (2012)
[22] Moradi, A., Mischke, O., Eisenbarth, T.: Correlation-enhanced power analysis collision attack. In: International Workshop on Cryptographic Hardware and Embedded Systems. pp. 125–139. Springer (2010)
[23] Nikova, S., Rechberger, C., Rijmen, V.: Threshold implementations against side-channel attacks and glitches. In: International conference on information and communications security. pp. 529–545. Springer (2006)
[24] Reparaz, O., Bilgin, B., Nikova, S., Gierlichs, B., Verbauwhede, I.: Consolidating masking schemes. In: Annual Cryptology Conference. pp. 764–783. Springer (2015)
[25] Secure-IC: CATALYZR tool (CTZ), https://www.secure-ic.com/solutions/catalyzr/, https://www.secure-ic.com/solutions/catalyzr/ and https://cadforassurance.org/tools/design-for-trust/catalyzr/, Accessed online on March 4th, 2021
[26] Soydan, S.: Analyzing the DPA leakage of the masked s-box via digital simulation and reducing the leakage by inserting delay cells. In: 2010 Fourth International Conference on Emerging Security Information, Systems and Technologies. pp. 221–227. IEEE (2010)
[27] Trichina, E.: Combinational Logic Design for AES SubByte Transformation on Masked Data. IACR Cryptology ePrint Archive **2003**, 236 (2003)
[28] Wild, A., Moradi, A., Güneysu, T.: Glifred: Glitch-free duplication towards power-equalized circuits on fpgas. IEEE Transactions on Computers **67**(3), 375–387 (2017)
[29] Xiao, G.Z., Massey, J.L.: A spectral characterization of correlation-immune combining functions. IEEE Transactions on information theory **34**(3), 569–571 (1988)

APPENDIX

A. More details on our design

In fig. 9, we illustrate the result of the synthesis of the masked GF_{16} inverter, which implements our design given in equation eq. (5). We implemented in Verilog our novel Boolean equations for the inversion in GF_{2^4}. The nets not to be simplified have been constrained to be kept in the netlist. Otherwise, we let the synthesizer (Cadence Encounter) optimize the netlist by merging common sub-expressions. The resulting netlist is displayed in Fig. 9. We have verified formally that each node fulfils the requirements of corollary 1 and proposition 1. The combinational gates are as usual, and rectangle symbols represent the 24 DFFs.

For the full AES, fig. 10 shows our implementation of the S-Box. The names of the signals are the same as the original one from [8].

- $A = (a_3, a_2, a_1, a_0, b_3, b_2, b_1, b_0)$: 8-bits masked input
- $M = (m_3, m_2, m_1, m_0, n_3, n_2, n_1, n_0)$: 8-bits input mask
- N: 8-bits output mask

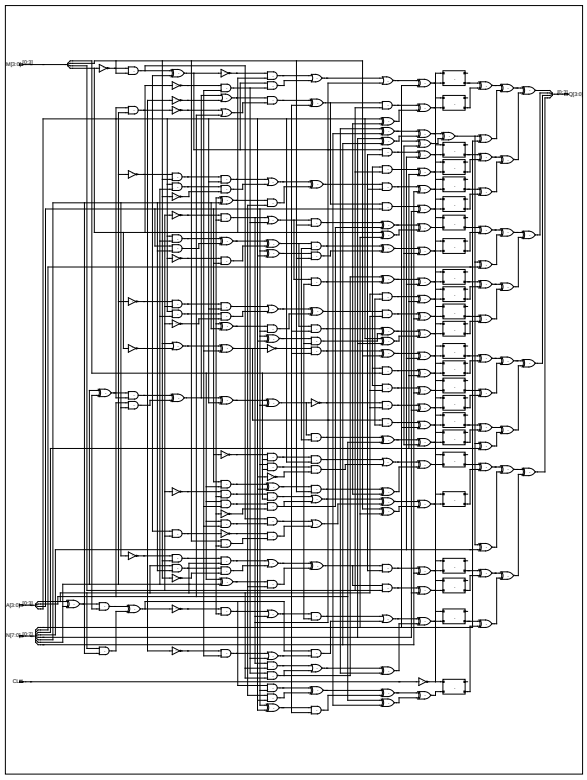


Fig. 9: Masked GF_{2^4} inverter synthesis result. The synthesizer did not optimize the design. All signals are correctly kept (using "keep" attribute).

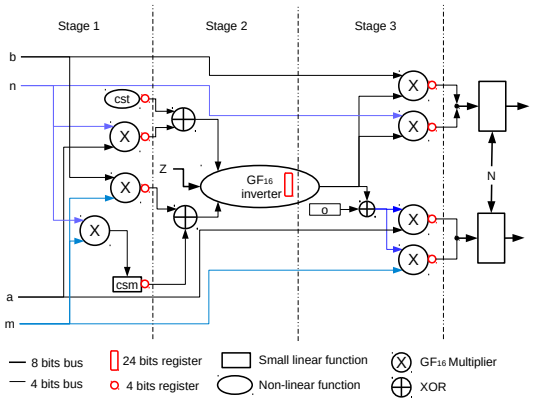


Fig. 10: AES S-Box scheme using our secure GF_{2^4} inverter.

- $Z = (N, z_1, \cdots, z_{18})$: 26 fresh random bits including N

The stage 1 and 3 are identical to the stage 1 and stage 4 of the DOM S-Box. The different output bits of the inverter GF_{16} are xored together at the GF_{16} multipliers level. We therefore masked the 24 FFs with different masks to avoid any transition resulting from a delayed register with an identical mask. Indeed we need 18 fresh random bits ($Z = (N, z_1, \cdots, z_{18})$) .