# Consolidating Masking Schemes*

Oscar Reparaz, Begül Bilgin,
Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede
firstname.lastname@esat.kuleuven.be

KU Leuven ESAT/COSIC and iMinds, Belgium

**Abstract.** In this paper we investigate relations between several masking schemes. We show that the Ishai–Sahai–Wagner private circuits construction is closely related to Threshold Implementations and the Trichina gate. The implications of this observation are manifold. We point out a higher-order weakness in higher-order Threshold Implementations, suggest a mitigation and provide new sharings that use a lower number of input shares.

**Keywords:** Masking, Private Circuits, Ishai–Sahai–Wagner, Threshold Implementations, Trichina gate, higher-order DPA

## 1 Introduction

Side-channel cryptanalysis allows to break implementations of mathematically secure cryptographic algorithms running on embedded devices. Shortly after the introduction of a particularly powerful branch of side-channel attacks, namely Differential Power Analysis (DPA) by Kocher et al. [15], different countermeasures were proposed. An especially popular countermeasure used today is masking, introduced in [7,12], mainly due to its theoretical soundness. Contrary to other heuristic, ad-hoc approaches, masking carries a proof of security [7]. A $d^{\text{th}}$-order secure masking works by splitting every sensitive variable (i.e. that depends on the key) into $s$ shares, such that an adversary probing at most $d$ values during the computation gets no information about any sensitive variable. This adversarial model is relevant in practice since the adversary is not weaker than a $d^{\text{th}}$-order DPA attack [11,8]. The advantage of a properly masked implementation is that it forces the adversary to use higher-order DPA attacks in order to break it. Higher-order DPA attacks are substantially harder to launch, both in terms of data complexity and computational resources [7,19,24]. Masking, however, comes with a cost. Performing operations in the masked domain increases the computational requirements on the target platform (area, time and randomness, among others), thus in practice, it is crucial to design countermeasures that have a limited cost impact.

In this paper, we focus on Boolean masking, i.e. the intermediates are split additively in a given finite field. The difficulty is then reduced to masking functions that are not linear with respect to addition.

### 1.1 Related works

There have been several efforts for constructing masking schemes —algorithms to compute on masked data. Some early development came from practitioners which produced designs mostly oriented to fit in real-world constraints: Trichina presents in [29] a masked AND gate resistant to first-order DPA attacks (first-order secure masking).

A generic algorithm for the masked AND computation at any security level is given by Ishai, Sahai and Wagner (here ISW) in [14], together with a convenient theoretical framework to prove the security of such a scheme. It is, however, well known that early theoretical concepts of masking schemes rely on assumptions that do not necessarily hold in practice. This is true for both the hardware and the software side. A common problem for the latter is that Hamming distance leakage, which is typically visible in memory-element transitions, may invalidate the assumption that leakages from each share are independent [1]. For the former, glitches (in static CMOS, a spurious transition of nodes in a combinational circuit within one clock cycle, resulting from different arrival times of the input signals) were shown to be a source of exploitable leakage [17,18], enabling successful DPA attacks against theoretically sound masked implementations due to unsatisfactory leakage modeling.

The mitigation of glitches is a well-studied problem in digital design, since they are not only inconvenient from a security point of view. Glitches are useless transitions that consume extra energy and thus digital designers tend to minimize them to achieve low-power and high-speed circuits. There are strategies to reduce glitches (e.g., balancing the path delay using combinational tree-like structures) or fully eliminate them (e.g. using dynamic logic styles, such as Domino or dynamic differential such as SABL [27] or WDDL [28]).

Alternatively, a specific strand of masking schemes, namely Threshold Implementations (TIs), were introduced in [21] to address the aforementioned model limitations. TIs are designed to deal with non-idealities in hardware (glitches) at a higher level of abstraction, and can provide strong security guarantees that may be relevant in practice. While ISW requires first to decompose a circuit into (exclusively) AND, XOR and NOT gates and then masking those, TI has the advantage that any function can be shared directly, which typically results in more compact designs. Recently TIs were extended to provide not only first-order but also higher-order security [3].

### 1.2 Our contribution

The discussion provided in this paper is threefold. First, we point out the similarities and differences between ISW, TI and the Trichina gate when the function to mask is an AND gate. We gain deeper understanding about masking schemes from these relations and use it to provide a generalized masking scheme (Section 3).

In the second part of the paper, we show how this generalization is mutually beneficial to all three masking schemes mentioned above. We show a weakness in the recently proposed higher-order extension of TI and suggest a fix using ideas from the generalized scheme in Section 4. In addition, we discuss how ISW and the Trichina masked AND-gate can be implemented securely in logic styles that glitch.

Finally, we focus on constructive applications. In Section 5.1, we discuss under which conditions a TI function provides security against $d^{\text{th}}$-order attacks using only $d+1$ shares instead of the usual $td+1$ bound. We end the paper by describing how ideas from TI could be inherited in software-oriented schemes to provide security in a distance-based leakage model (Section 5.2).

## 2 Preliminaries

We begin with standard definitions and descriptions of the masking schemes that we consider. Lower-case characters refer to elements in a field with characteristic two. An element $a$ is split into $s$ shares $a_i$, where $i \in 1, 2, \ldots, s$ by means of Boolean masking. Namely, without loss of generality $s-1$ random shares $a_1, \ldots, a_{s-1}$ are drawn from the uniform distribution, then $a_s$ is calculated such that $a = \bigoplus_s a_i$. Bold characters refer to a valid shared vector $\mathbf{a} = a_1, \ldots, a_s$. We use the term $s$-share representation ($s$-sharing) of $a$ to emphasize the number of shares. Note that the sharing

**a** generated as detailed above is uniform [4]. Moreover, the sharing $\mathbf{a}_{\bar{\mathbf{i}}} = a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_s$ is independent of the unshared value $a$ for any choice of $i$. It is hence an $(s, s)$ secret sharing.

We use upper-case characters to denote functions. For a given unshared function $b = F(a)$, we generate a shared vector $\mathbf{F} = F_1, \ldots, F_s$ of component functions $F_i$ in order to perform the shared calculation. The sharing $\mathbf{F}$ is *correct* if $b = \bigoplus_s b_i$ for $b_i = F_i(\mathbf{a})$. The algebraic degree of a function is denoted with $t$.

*Adversarial model.* We use $d$ probing as our adversarial model which we define as follows. The adversary is allowed to probe $d$ wires in the circuit within a certain time window. Each probed wire $g$ calculating a function $G$ gives information about all the inputs of $G$ up to the latest synchronization point[1]. This definition directly implies that the adversary can derive all the intermediate values during the computation of $G$ and hence the output of $G$. To clarify, let us refer to Figure 1. An attacker probing the output of the function $G$ (that is, wire $g$) can observe all the inputs to $G$ up to, and including, $\text{reg}_2$; can generate all the intermediate values used during the calculation of $G$ (including the outputs of $G$ that are stored in $\text{reg}_3$); but can not directly learn all the values stored in $\text{reg}_1$ or any intermediate values occurring during the calculations of each $F_i$.

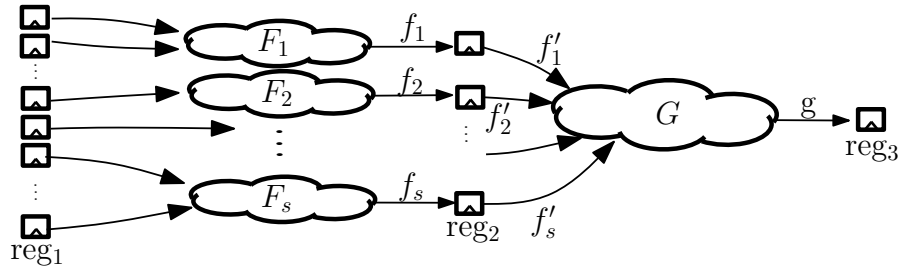


Fig. 1: Exemplary circuit to aid the explanation of the adversarial model adopted in this paper. $F_i$ and $G$ are combinational logic blocks, $\text{reg}_i$ are register stages and $f_i$ and $g$ are wires that compute the $F_i$ (resp. $G$) functions.

We note that this is a theoretical model stronger than a real-world attacker since a real-world attacker can only get a subset of the mentioned information. Moreover, it is slightly different than the conventional $d$-probing model [14]. Nevertheless, it is advantageous since being able to see the inputs of the gates used during the calculation implies the ability to observe real world effects such as glitches. Hence, we gain the flexibility to work also with non-ideal (glitchy) gates. If the usage of ideal gates is assumed, working with the conventional model is typically sufficient.

This model matches quite nicely with $d^{\text{th}}$-order DPA attacks, which consider a noisy function of intermediates' leakage [11]. There are certainly other adversarial models that are even more powerful, in which the attacker has the ability to adaptively move the probes between time periods (but not within a time period). We note that this "adaptive-probes" model is stronger and we do not consider moving probes in this paper.

*Ishai–Sahai–Wagner scheme.* Private circuits [14] provide a procedure for computation on masked data. They give a construction for NOT and AND gates, and prove the security against $d$ probes of any circuit composed of these secure gates ("gadgets") which are in turn built from logic

[1] One of the many ways of synchronization is storing elements in registers which we inherit throughout this paper without loss of generalization.

**Require:** $s$-shares **a** and **b**
**Ensure:** $s$-shares **c** satisfying $c = ab$
  **for** $i$ from 1 to $s$ **do**
    **for** $j$ from $i+1$ to $s$ **do**
      $z_{ij} \leftarrow$ `rnd()`
      $z_{ji} \leftarrow (z_{ij} \oplus a_i b_j) \oplus a_j b_i$
    **end for**
  **end for**
  **for** $i$ from 1 to $s$ **do**
    $c_i \leftarrow a_i b_i$
    **for** $j$ from 1 to $s$, $j \neq i$ **do**
      $c_i \leftarrow c_i \oplus z_{ij}$
    **end for**
  **end for**

Fig. 2: ISW algorithm.

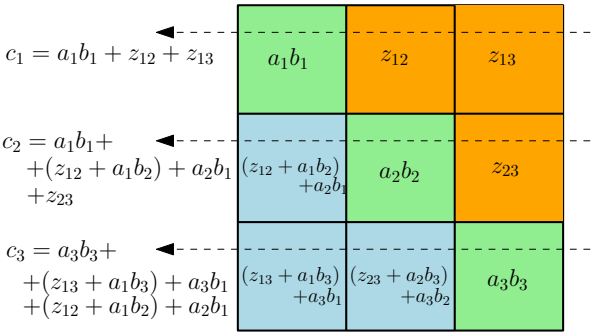


Fig. 3: Intermediate state of the ISW computation for $s = 3$.

gates that do not glitch. To compute $c = F(a, b) = ab$ while providing security against $d$ probes, ISW takes $s = 2d + 1$ shares **a** and **b** of each input and consumes $\binom{s}{2}$ bits of randomness. We exemplify the computation of a masked AND gate providing security against adversaries using one ($d = 1, s = 3$) and two ($d = 2, s = 5$) probes in Equation (1), and Equations (2) and (3) respectively. An intermediate state of the computation is shown in Figure 3.

$$\begin{aligned} z_{21} &= (z_{12} \oplus a_1b_2) \oplus a_2b_1, & c_1 &= a_1b_1 \oplus z_{12} \oplus z_{13}, \\ z_{31} &= (z_{13} \oplus a_1b_3) \oplus a_3b_1, & c_2 &= a_2b_2 \oplus z_{21} \oplus z_{23}, \\ z_{32} &= (z_{23} \oplus a_2b_3) \oplus a_3b_2, & c_3 &= a_3b_3 \oplus z_{31} \oplus z_{32}. \end{aligned} \tag{1}$$

First, three (resp. ten) bits of randomness $z_{ij}$ where $1 \leq i < j \leq s$ are drawn i.i.d. uniformly random. Then the intermediates $z_{ji}$ are computed as shown in the left column of Equation (1) (resp. Eqn. (2)). The last step xors the intermediates $z_{ij}$ and the products $a_ib_i$ to compute the $s$ output shares **c** (right column of Eqn. (1) and Eqn. (3) respectively).

$$\begin{aligned} z_{21} &= (z_{12} \oplus a_1b_2) \oplus a_2b_1, & z_{31} &= (z_{13} \oplus a_1b_3) \oplus a_3b_1, \\ z_{41} &= (z_{14} \oplus a_1b_4) \oplus a_4b_1, & z_{51} &= (z_{15} \oplus a_1b_5) \oplus a_5b_1, \\ z_{32} &= (z_{23} \oplus a_2b_3) \oplus a_3b_2, & z_{42} &= (z_{24} \oplus a_2b_4) \oplus a_4b_2, \\ z_{52} &= (z_{25} \oplus a_2b_5) \oplus a_5b_2, & z_{43} &= (z_{34} \oplus a_3b_4) \oplus a_4b_3, \\ z_{53} &= (z_{35} \oplus a_3b_5) \oplus a_5b_3, & z_{54} &= (z_{45} \oplus a_4b_5) \oplus a_5b_4. \end{aligned} \tag{2}$$

$$\begin{aligned} c_1 &= a_1b_1 \oplus z_{12} \oplus z_{13} \oplus z_{14} \oplus z_{15}, & c_4 &= a_4b_4 \oplus z_{41} \oplus z_{42} \oplus z_{43} \oplus z_{45}, \\ c_2 &= a_2b_2 \oplus z_{21} \oplus z_{23} \oplus z_{24} \oplus z_{25}, & c_5 &= a_5b_5 \oplus z_{51} \oplus z_{52} \oplus z_{53} \oplus z_{54}. \\ c_3 &= a_3b_3 \oplus z_{31} \oplus z_{32} \oplus z_{34} \oplus z_{35}, \end{aligned} \tag{3}$$

Extensions to higher orders are similarly generated using the algorithm in Figure 2.

It is well known that the ISW algorithm can work in larger finite fields by building upon field multiplications instead of AND gates. In the case of AES, there is a significant performance gain if ISW operates in GF($2^8$), due to the algebraic structure of the AES S-box [25]. We refer to [14] for a variant of this method using $s = d + 1$ shares.

*Threshold Implementations.* TI provides provable security against $d^{\text{th}}$-order DPA even in a circuit with glitches according to [3]. In addition, it is also advantageous since any degree $t$ function can be securely implemented using at least $s \geq td + 1$ shares.

The security of a single function relies on the satisfaction of *$d^{th}$-order non-completeness*: any combination of up to $d$ component functions $F_i$ of $\mathbf{F}$ must be independent of at least one input share. It is shown that such a sharing can always be constructed using $s_{in} = td + 1$ and $s_{out} = \binom{s_{in}}{t}$ shares. Examples for the function $d = F(a, b, c) = c \oplus ab$ are given in Equations (4) and (5) for $d = 1$ and $d = 2$ respectively.

$$\begin{aligned} d_1 &= c_2 \oplus a_2b_2 \oplus a_1b_2 \oplus a_2b_1, \\ d_2 &= c_3 \oplus a_3b_3 \oplus a_3b_2 \oplus a_2b_3 \\ d_3 &= c_1 \oplus a_1b_1 \oplus a_1b_3 \oplus a_3b_1. \end{aligned} \tag{4}$$

Notice that $s_{out} > s_{in}$ for $d > 1$. In order to avoid further increase of the number of shares when several functions are cascaded, some of the output shares are typically xored. It is important that this reduction is performed only after the $s_{out}$-sharing $\mathbf{d}$ is stored in the registers in order to satisfy the non-completeness property and to avoid glitches depending on all shares of a variable.

$$\begin{aligned} d_1 &= c_2 \oplus a_2b_2 \oplus a_1b_2 \oplus a_2b_1, & d_2 &= c_3 \oplus a_3b_3 \oplus a_1b_3 \oplus a_3b_1, \\ d_3 &= c_4 \oplus a_4b_4 \oplus a_1b_4 \oplus a_4b_1, & d_4 &= c_1 \oplus a_1b_1 \oplus a_1b_5 \oplus a_5b_1, \\ d_5 &= a_2b_3 \oplus a_3b_2, & d_6 &= a_2b_4 \oplus a_4b_2, \\ d_7 &= c_5 \oplus a_5b_5 \oplus a_2b_5 \oplus a_5b_2, & d_8 &= a_3b_4 \oplus a_4b_3, \\ d_9 &= a_3b_5 \oplus a_5b_3, & d_{10} &= a_4b_5 \oplus a_5b_4. \end{aligned} \tag{5}$$

In order to provide security when several functions are cascaded, (i.e. the output of $\mathbf{F}$ is used as the input to another shared nonlinear-function $\mathbf{G}$), the shared function and its output should satisfy *uniformity* [4]. Several methods to achieve uniformity have been proposed [5,6,16,22]. It is advised to use re-masking [4,20] in case these methods do not provide a solution.

When a single AND gate is considered, it has been shown that there exists no 3-sharing satisfying both uniformity and first-order non-completeness [5]. Therefore, the output shares of the 3-share AND gate must be re-masked (refreshed). Moreover, the sharing in Equation (4) considering an AND and XOR gate instead of a single AND gate satisfies all TI properties.

*Trichina AND-gate.* Unlike ISW and TI which can be applied both at the algorithm and at the gate level, Trichina [29] investigates how to implement a masked AND gate $c = ab$ securely strictly at the gate level. The construction, which is described in Equation (6), requires two 2-share inputs and uses 1-bit extra randomness $z_{12}$ to generate a 2-share output. The security relies strictly on the order of the operations to avoid unmasking certain bits, on the ideality of the cells and on the assumption that the sharing $\mathbf{a}$ of $a$ is independent from $\mathbf{b}$.

$$\begin{aligned} c_1 &= (((z_{12} \oplus a_1b_2) \oplus a_2b_1) \oplus a_2b_2) \oplus a_1b_1 \\ c_2 &= z_{12}. \end{aligned} \tag{6}$$

## 3 Conciliation

In this section we first relate the ISW scheme with TI and the Trichina gate using elementary transformations. We then use ingredients from all three schemes to describe a generalized masking construction and argue its security. As a case study, we consider a first-order sharing of an AND gate.

### 3.1 From ISW to TI

Consider the ISW construction with $s = 3$ input shares, providing first-order security as depicted in Figure 4. It is equivalent to the computation in Equation (1) and to Figure 2. In Figure 4, the computation flows from the outside towards the center. It begins with deriving all the cross products $a_i b_j$. Then three random values $z_{ij}$ are added to some of the cross products. The terms are finally xored together in three groups to generate the output shares $c_i$.

In the following, we perform several elementary transformations on this circuit to arrive to a typical re-masked 3-share TI of an AND gate.

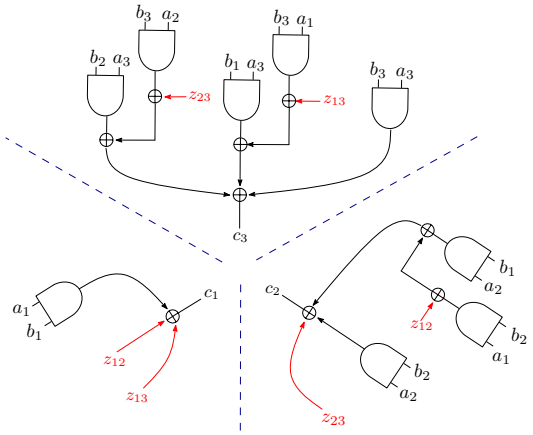


Fig. 4: Original ISW scheme.

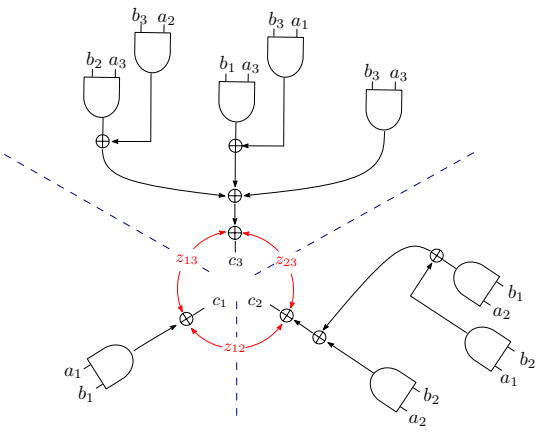


Fig. 5: After first transformation.

*First transformation: moving random bits.* Delaying the injection of randomness using the random bits $z_{ij}$ closer to the center (towards the end of the calculation) as depicted in red yields the construction in Figure 5. Of course, this operation preserves the correctness of the output. It is already possible to recognize a refreshing operation in the inner ring where $z_{12}$, $z_{13}$ and $z_{23}$ are involved. Note that the security of this intermediate construction highly depends on the order of computation of the XOR gates and the ideality (glitching behavior) of the gates.

*Second transformation: moving AND gates.* The next modification transforms the circuit of Figure 6 into Figure 7. It simply moves around the two red AND gates $a_1 b_3$ and $a_3 b_1$ together with the XOR gate from the upper to the lower-left branch.

This second transformation also preserves the correctness at the output. Notice that after this transformation each branch of the circuit sees at most two shares of each input. For example, the upper branch sees only $a_2$, $a_3$, $b_2$ and $b_3$. We can absorb the computation from each branch (3 ANDs and 2 XORs) into its respective component function $F_i$ as shown in Equation (7).

$$\begin{aligned} F_1(a_1, a_2, b_1, b_2) &= a_2 b_2 \oplus a_1 b_2 \oplus a_2 b_1, \\ F_2(a_2, a_3, b_2, b_3) &= a_3 b_3 \oplus a_2 b_3 \oplus a_3 b_2, \\ F_3(a_1, a_3, b_1, b_3) &= a_1 b_1 \oplus a_3 b_1 \oplus a_1 b_3. \end{aligned} \tag{7}$$

The reader will recognize that the resulting sharing $\mathbf{F}$ is a TI (satisfying first-order non-completeness) followed by a refreshing (resulting from the first transformation.) Note that one

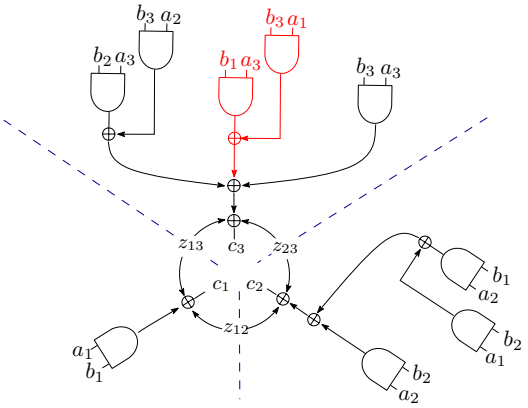


Fig. 6: Before second transformation.

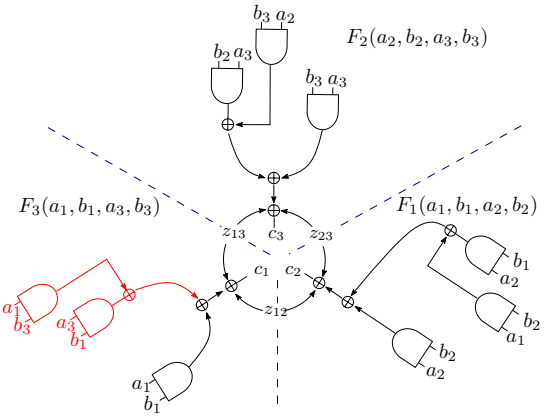


Fig. 7: After second transformation.

could also equivalently see this refreshing as an addition with the uniform shares of the constant value 0.

The security of this construction follows from the fact that it is a TI, followed by a refreshing. In particular, this construction is secure even in the presence of glitches. Therefore, we link the $s = 3$ ISW scheme to first-order TI.

### 3.2 From ISW to the Trichina AND-gate

In Figure 8, we draw the ISW computation[2] of an AND gate for $s = 2$. In Figure 9, we have the Trichina AND gate. Similar to Section 3.1, we can transform the ISW construction $s = 2$ to the Trichina gate by rearranging the term $a_1 b_1$. It is noteworthy that the Trichina gate resembles a simplified ISW, and thus can be seen as the practitioners version.

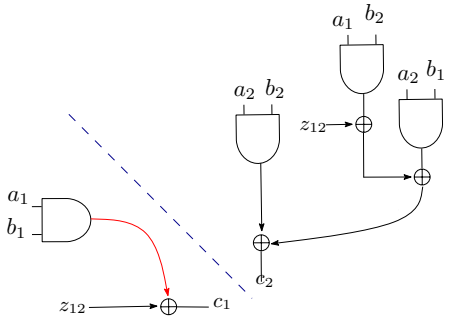


Fig. 8: ISW with $s = 2$.

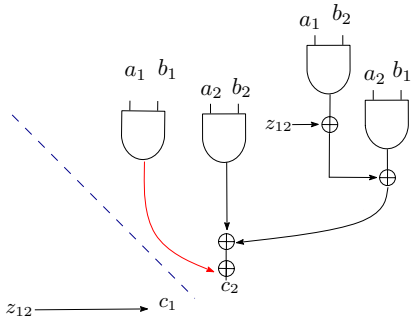


Fig. 9: Trichina AND gate.

[2] We stress that ISW with $s = 2$ is neither strictly defined nor proven secure in the ISW simulation model. We are extending the algorithm in Figure 2 in a straight forward way to any $s$.

### 3.3 Generalizing and inducing a structure

We can generalize the masked AND-gate transformations from Sections 3.1 and 3.2 to the general case of higher orders. In addition, we can construct variants that compute logic gates with more than two inputs or more sophisticated functions.

In order to induce a structure to the mentioned generalization, we decompose the resulting construction into four layers as exemplified in Figures 10 and 11 for first- and second-order security respectively. Specifically, we notice a non-linear layer $\mathcal{N}$, followed by a linear layer $\mathcal{L}$ and a refreshing layer $\mathcal{R}$. In certain cases where we want to reduce the number of shares, we add a linear compression layer $\mathcal{C}$. Below we detail the functionality of each layer.

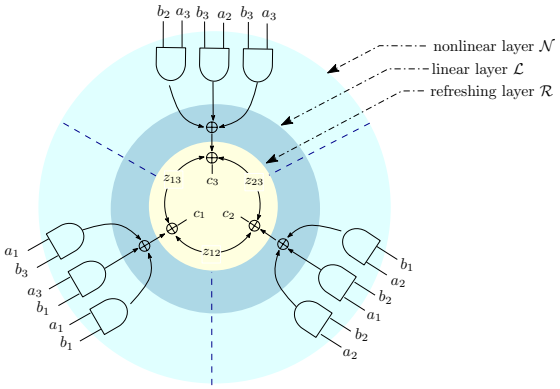


Fig. 10: First-order secure ($s = 3$) after transformation.

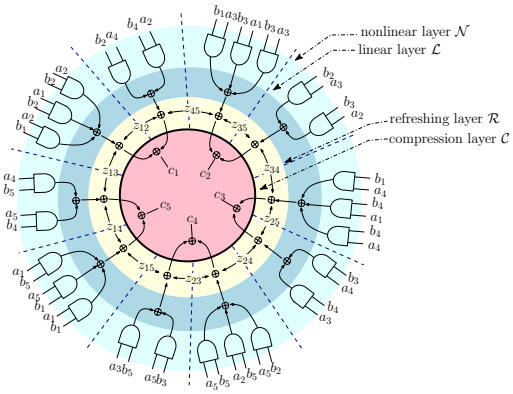


Fig. 11: Second-order secure ($s = 5$) after transformation.

*The non-linear layer $\mathcal{N}$.* This layer is responsible for the bulk of the computation, corresponding to the cross products $a_i b_j$. For example, in the first-order secure construction of a 3-share 2-input AND gate, $\mathcal{N}(\mathbf{a}, \mathbf{b}) = (a_1 b_1, a_1 b_2, \ldots, a_3 b_3)$ maps 6 input bits to 9 output bits ($a_i b_j$). Note that the set of cross products calculated in this layer is defined by the number of shares and the function itself. [3]

In order to generalize the construction such that a function other than $c = ab$ is computed (such as $d = abc$, $d = a \oplus bc$, $d = a \oplus bc \oplus abc$), $\mathcal{N}$ needs to be modified accordingly. To be specific, all the shared terms (cross products and linear terms) should be placed in $\mathcal{N}$ to be used in the following steps.

*The linear layer $\mathcal{L}$.* This layer is an XOR net that reduces the number of shares without modifying the unshared value. In the AND-gate example, it maps 9 input bits (output of $\mathcal{N}$) to 3 output bits for the first-order case and 25 input bits to 10 output bits in the second-order case. The linear layer $\mathcal{L}$ of TI is responsible for preserving non-completeness. Failure to achieve non-completeness can cause sensitive information leakage in a glitchy circuit as in the case of the original ISW scheme (see Section 4.2).

The reduction of the number of shares performed by $\mathcal{L}$ partially limits the exponential blow-up of shares otherwise caused by the non-linear layer $\mathcal{N}$ alone. We point out that the output of $\mathcal{N}$ is

[3] It is possible to add other terms to $\mathcal{N}$ (as long as they are inserted an even number of times), such as virtual variable pairs [6], in order to increase the flexibility for generating a uniform sharing.

already a valid, non-complete sharing when each cross product is considered as an output share. However, cascading several sharings without $\mathcal{L}$ increases the number of shares rapidly, making such an implementation impractical except for circuits with very shallow logic depth.

*The refreshing layer $\mathcal{R}$.* This layer is applied in order to re-mask the output of $\mathcal{L}$. It is shown in several prior works on first-order TI that this layer can be avoided if the output of $\mathcal{L}$ already satisfies uniformity. However, $\mathcal{R}$ is critical in order to provide higher-order security as will be discussed in Section 4.1. In ISW, each output of each masked AND gate is refreshed, which clearly increases the randomness requirements compared to the generalized sharing of a more complex function with several terms (such as $d = a \oplus bc \oplus abc$).

*The compression layer $\mathcal{C}$.* While designing the $\mathcal{L}$ layer, there is a natural tension between two desirable properties, namely satisfying $d^{\text{th}}$-order non-completeness and having a small number of output shares. Normally, one designs $\mathcal{L}$ to have a small number of output shares yet satisfying $d^{\text{th}}$-order non-completeness. One example of this issue is the second-order masking of an AND-gate depicted in Figure 11, where the number of shares at the output of $\mathcal{L}$ (10 shares) is considerable larger than the number of shares of each input variable (5-share $\mathbf{a}$ and $\mathbf{b}$).

If it is desired to decrease the number of output shares further, the compression layer $\mathcal{C}$ is applied. This layer is composed of XOR gates only. In order to satisfy non-completeness and avoid glitches causing leakage of more than the intended number of shares, it is crucial to isolate the $\mathcal{R}$ and $\mathcal{C}$ layers using registers.

Note that in typical TIs, the layers $\mathcal{N}$ and $\mathcal{L}$ are combined and absorbed into the component functions without registers between these layers as drawn in Figure 11. An additional challenge is to design $\mathcal{L}$ so that it simultaneously satisfies non-completeness and uniformity.

### 3.4 Security arguments for generalized scheme

In this section, we argue the security of the generalized structure. We start by showing the security of a 2-input AND gate (2AND) against a $d$-probing adversary and inductively continue to a function of degree $t$. We assume that inputs to $\mathcal{N}$ are uniformly shared and synchronous. This discussion enables us to relate the number of required shares in TI with that in ISW.

*2-input AND gate.* Let us consider a set of information $I$ (based on indices) gathered by the attacker by probing $d$ wires. Specifically, if a wire corresponding to $a_i, b_i$ or $a_i b_i$ is probed, the index $i \in I$. If the wire corresponding to $a_i b_j$ is probed, both $i, j \in I$. This implies that a probed wire at the output of the layer $\mathcal{N}$ can give information about at most two indices. Therefore, the cardinality of $I$ is at most $2d$ when $d$ wires are probed in $\mathcal{N}$. It follows that using at least $2d + 1$ shares is required to provide security up to $\mathcal{L}$. However, an attacker is not limited to probing certain layers. Notice that the attacker probing closer to the end of the calculation of the component functions, i.e. just before the register between $\mathcal{R}$ and $\mathcal{C}$, gains more information. By the definition of the linear layer $\mathcal{L}$, the component functions should be formed such that any combination of up to $d$ of them should be independent of at least one share, i.e. one index, when a $d$-probing secure circuit is considered. Hence, we know that if it is possible to construct $\mathcal{L}$ with the given restriction, the attacker probing $d$ wires never has all the indexes in $I$. Since the input shares are uniformly shared and the vectors $\mathbf{a}_{\bar{\mathbf{i}}}, \mathbf{b}_{\bar{\mathbf{i}}}, \ldots$ are independent from the unshared values $a, b, \ldots$, we achieve security at the end of $\mathcal{L}$. Moreover, knowing the randomness used in $\mathcal{R}$ does not yield additional information to the attacker. At this point the possibility of generating $\mathcal{L}$ with $2d + 1$ shares becomes the question. It has been shown in [3] with a combinatorial argument that this is possible if the linear layer $\mathcal{L}$ is divided into $\binom{2d+1}{2}$ component functions. Namely, each

component function uses at most two input shares and hence at most two indices are put in $I$ for each probing. This gives the cardinality of at most $2d$ when $d$ probes are used.

Notice that the security discussion provided so far considers only one AND gate. However, the security of the generalized scheme also holds for the composition of several AND gates. Namely, the refreshing layer $\mathcal{R}$ and the register afterwards impose independence of the composed elements and non-completeness respectively. Hence, the union of the gathered information does not give an additional advantage to the attacker.

In the case of a single-probe adversary, we can relax the requirements on $\mathcal{R}$. As long as the next nonlinear function sees uniformly shared inputs, one can simplify the construction of $\mathcal{R}$ and even in some cases avoid $\mathcal{R}$. This result follows from the fact that an attacker using a single probe is unable to combine information from more than one function.

*3-input AND gate.* The security argument for a 3-input AND gate ($F(a, b, c) = abc$) follows the same lines as for the 2-input AND gate. The nonlinear layer $\mathcal{N}$ calculates $a_i b_j c_k$ terms. In order to keep the number of shares small, we need to make sure that each component function uses variables with at most 3 different indices. Then, an attacker probing $d$ wires can only gather information from at most $3d$ indices. The question if it is possible to arrange $\mathcal{L}$ such that this restriction is respected is answered positively in [3]. It can be done by dividing $\mathcal{L}$ into $\binom{3d+1}{3}$ component functions. Note that for a full proof of security, the insertion of randomness (the $\mathcal{R}$ layer) and registers become critical in order to provide higher-order security of the composition of such gates.

Naturally, it is also possible to generate a shared 3AND gate by composing two shared 2AND gates. This requires usage of registers after both the first and the second 2AND-gate calculation. However, the construction described above which performs the 3AND gate calculation at once is typically more efficient.

*$t$-input AND gate.* We can inductively apply the arguments for 2AND and 3AND gates to the $t$-input AND gate. This implies the sufficient lower bound of $s_{in} = td + 1$ input shares. The shared function should be split into at least $\binom{s_{in}}{t}$ component functions in $\mathcal{L}$ and satisfy $d^{\text{th}}$-order non-completeness.

*Degree $t$ Boolean functions.* The above inductive argument does not exclude functions composed of more than one degree $t$ term. To clarify, the generation of $\mathcal{N}$ is performed by straight-forward calculation of all cross products using $s_{in} = td + 1$ shares. The linear layer is split into $\binom{s_{in}}{t}$ component functions, each of which sees $t$ indices as described above for a $t$-input AND gate. Any shared term of degree $\leq t$ can be placed to at least one existing component function since any cross product of the shared $\leq t$ term uses at most $t$ indices, which concludes the argument.

*Degree $t$ functions in other finite fields.* A careful investigation of the above arguments shows that they are independent of the used field. Namely, it is enough to replace the AND gates in GF(2) with multiplication in the required field in order to provide security for a degree $t$ function.

We conclude the security argument of the generalized masking scheme by noting that $s_{in}$ can be chosen to be greater than $td + 1$ in order to achieve flexibility without invalidating the security arguments.

### 3.5 Wrapping up.

In this section, we provided a generalized scheme which extends ISW and TI like masking schemes. Specifically, unlike the ISW scheme which builds up on AND gates or field multiplications; the generalized scheme allows to implement any function directly, enabling the usage of less

compositions. The generalized scheme inherits the ability to operate on larger fields and security against $d$-probing adversary. In addition, it offers protection for composition of gates.

## 4 What can go wrong?

In Section 3 we constructed a generalized masking structure, and assigned precise requirements and functions to each of its layer. In this section, we show how small deviations from this generalized scheme can cause vulnerable implementations. In particular, in Section 4.1 we analyze the cost of lacking a refreshing layer $\mathcal{R}$. We use the recently proposed higher-order TI as our case study to show a higher-order flaw, then we suggest a generic fix. In Section 4.2, we elaborate on the insecurity that deviating from the structure especially on $\mathcal{L}$ brings in the presence of glitches using the ISW and Trichina scheme as our case study.

### 4.1 Higher-order TI is not so higher-order secure

The higher-order TI proposed in [3] fits to our generalized structure as follows. $\mathcal{N}$ and $\mathcal{L}$ together enforce a correct and $d^{\text{th}}$-order non-complete implementation. However, unlike the generalized scheme, the refreshing layer $\mathcal{R}$ is not performed in TI when the uniformity of the shared output of $\mathcal{C}$ can be satisfied without $\mathcal{R}$. This difference becomes important since as we shall see in the sequel, it can induce a higher-order security flaw when composing several sharings, even if these sharings are uniform.

For simplicity, we use a second-order TI of a *mini-cipher* construction and show a second-order leakage.

*Description of the mini-cipher.* Let us consider a minimal non-linear feedback shift register. This mini-cipher comes from an extreme simplification of the KATAN block cipher for which a higher-order TI was given in [3]. We consider a 4-bit state $S[i]$, $i \in 0, \ldots, 3$ for which the taps are at the state bits with indices $i = 1, 2, 3$ and the feedback is plugged at position 0. This state is a "sensitive variable[4]". The feedback function ("round function" of an extremely unbalanced Feistel) $F = F(S[3], S[2], S[1])$ is the same AND-XOR feedback function as in KATAN, namely $d = F(a, b, c) = ab \oplus c$.

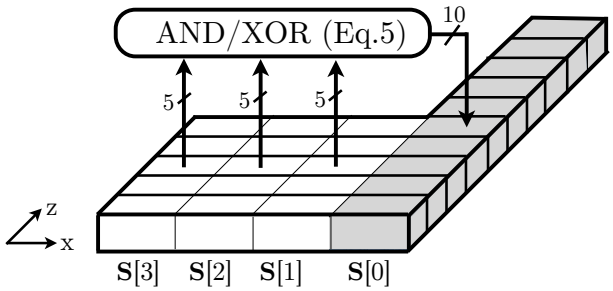


Fig. 12: Diagram of the shared version of the mini-cipher.

[4] The goal is to show leakage in this construction. To simplify and keep the essentials, we do not explicitly inject the key in this mini-cipher construction, but assume that the initial state is secret (sensitive).

*Shared version of the mini-cipher.* The shared version of this mini-cipher (non-complete sharing in the $\mathcal{N}$ and $\mathcal{L}$) follows the lines of [3]. The feedback function $F$ is shared as Equation (5). In particular, to provide security against glitches, the state bit $\mathbf{S}[0]$, in which the output of $\mathbf{F}$ is stored, is composed of 10 shares, whereas $\mathbf{S}[1], \mathbf{S}[2], \mathbf{S}[3]$ are composed of 5 shares. The conversion from 10 to 5 shares is done as suggested in [3]. That is, the fifth share of $\mathbf{S}[1]$ sees the xor of the last six shares of $\mathbf{S}[0]$ when the cipher is clocked.

*A second-order leakage.* Some lengthy, albeit straightforward, calculations show that the covariance (second mixed statistical moment) between the fifth share of $\mathbf{S}[1]$ after the first cycle and the fourth share of $\mathbf{S}[1]$ after the seventh cycle depends on the unshared initial value $S[2]$. Thus, there is a second-order flaw that invalidates the security claims of the scheme.[5]

*Mitigation.* The direct mitigation is to refresh the shares after each shared function computation, for example by adding fresh shares of the null vector. In other words, the refreshing layer $\mathcal{R}$ should be implemented in TI when higher-order security is considered. The idea here is to isolate the intermediates occurring within each computation stage from intermediates of another stage, so that combining intermediates from different stages no longer reveals information about a secret unshared value. With this argument, we fix the flaw in [3] using the conciliation of masking schemes.

Note that this fix naturally increases area and randomness requirements and we do not claim that it is the optimal solution. We foresee that this solution may be an overkill in many situations, and a careful analysis can save significant amount of randomness. This is especially true since the existence of a second-order dependency between two variables does not necessarily imply an easy key-extraction by DPA. In particular, if there is a second-order flaw between two intermediates for which there is enough key-diffusion between them, key recovery exploiting the joint leakage of intermediates becomes difficult. The exact minimum amount of $\mathcal{R}$ layers needed to make the whole implementation secure against higher-order attacks may depend from design to design.

### 4.2 ISW and Trichina in the presence of glitches

ISW scheme implicitly considers a logic gate that does not glitch. Thus, a straightforward translation of ISW into standard CMOS technology can result in a vulnerable implementation. To see this, observe that in Equation (8) $c_3$ breaks the non-completeness property:

$$
\begin{aligned}
c_1 &= a_1 b_1 \oplus z_{12} \oplus z_{13}, \\
c_2 &= a_2 b_2 \oplus ((z_{12} \oplus a_1 b_2) \oplus a_2 b_1)) \oplus z_{23}, \\
c_3 &= a_3 b_3 \oplus ((z_{13} \oplus a_1 b_3) \oplus a_3 b_1) \oplus (z_{23} \oplus a_2 b_3) \oplus a_3 b_2.
\end{aligned}
$$

The trivial fix here is to register signals that otherwise could result in undesired (and pernicious) glitches. More precisely, if during the ISW computation in logic, the intermediate values $z_{ji}$ where $i < j$ (outputs in Equation (1), left column; and Equation (2)) are stored in registers together with the intermediate values $a_i b_i$ before further XOR combinations, the circuit becomes secure even if there are glitches. This follows since non-completeness holds between register stages. The caveat of this fix is the significant increase in area (and latency) due to extra registers. Note that this extra layer of registers is prevented by careful selection of the layer $\mathcal{L}$.

Similar observations apply to the Trichina construction. Trichina also imposes restrictions on the logic gates, especially on the order of evaluation of these gates. It is implicitly assumed that signals are registered or latched in order to avoid glitches. The case where first-order security fails due to glitches is studied in [18].

[5] This result had been previously reported in [23] and experimentally confirmed in [26].

## 5 Applications

Here we introduce two additional constructive applications. In the first one, we focus on optimizing the generalized scheme further such that it uses less input shares per function. The second application analyses software-like implementations in a distance-based leakage model.

### 5.1 Using $d + 1$ input shares

As described in Section 3, the generalized scheme uses at least $td + 1$ input shares to protect a function with degree $t$ against $d$-probing attacks. Here, we improve the scheme such that it uses less input shares, specifically, $d + 1$ shares to achieve $d$-probing security. As a trade-off, however, this sharings are more restrictive with the requirements of *independent* input sharings. We illustrate with single-probe secure examples the design process of such sharings and the construction of layers. We provide a security argument and discuss connections with prior works.

*First crack.* We start with the first-order sharing of $c = ab$ with $s_{in} = 2$ and $s_{out} = 4$ given in Equation (8). The sharing $\mathbf{c}$ is only composed of the crossproducts $a_i b_j$. Hence, it can be seen as the output of $\mathcal{N}$ which is already a correct sharing for $c$. Moreover, if the sharing of $a$ is independent than that of $b$ then each share $c_i$ is independent of at least one input share of each variable. In other words, non-completeness is satisfied. This implies the independence of $\mathbf{c}$ from the unmasked variables $a$ and $b$ providing security under a single probe.

$$c_1 = a_1 b_1, \qquad c_2 = a_1 b_2, \qquad c_3 = a_2 b_1, \qquad c_4 = a_2 b_2. \tag{8}$$

Note that in this simple sharing, if the sharings of $a$ and $b$ were dependent (for example, $\mathbf{a} = \mathbf{b}$), then the second output share $a_1 b_2 = a_1 a_2$ would depend on all shares of $a$ (breaking non-completeness) and this would clearly leak information about $a$. During the construction of layers in the following, we assume that the sharings of each input variable is independent from all others.

*Construction of $\mathcal{N}$ and $\mathcal{L}$.* The increase of number of variables in the input increases the number of cross products and hence the number of output shares of $\mathcal{N}$ exponentially. For example, if we consider the sharing of $d = a \oplus ac \oplus bc$ with $s_{in} = 2$, we end up with 10 terms $(a_1, a_2, a_1c_1, a_1c_2, a_2c_1, a_2c_2, b_1c_1, b_1c_2, b_2c_1, b_2c_2)$ in $\mathcal{N}$. Notice that it is possible to reduce the number of output shares using a careful selection of a linear layer $\mathcal{L}$ as shown in Equation (9) while satisfying the non-completeness property.

$$\begin{aligned} d_1 &= a_1 \oplus a_1c_1 \oplus b_1c_1, & d_3 &= a_2 \oplus a_2c_1 \oplus b_2c_1, \\ d_2 &= a_1c_2 \oplus b_1c_2, & d_4 &= a_2c_2 \oplus b_2c_2. \end{aligned} \tag{9}$$

The number of output shares of $\mathcal{L}$ also changes significantly depending on the function itself in addition to the number of input shares. To clarify, let us consider the sharing of $d = a \oplus ac \oplus bc \oplus ab$ which differs from the prior unshared function in the additional term $ab$. The terms $(a_1b_1, a_1b_2, a_2b_1, a_2b_2)$ should be added to Equation (9) for a correct implementation. Even if we place the additional terms $a_1b_1$ and $a_2b_2$ to the first and the last component functions in Equation (9) respectively, the remainding terms $a_1b_2$ and $a_2b_1$ can not be placed in these four component functions without breaking the non-completeness property. Hence, we need to increase the number of shares. One option to obtain non-completeness is increasing the number of output shares of $\mathcal{L}$ as shown in the equation below.

$$\begin{aligned} d_1 &= a_1 \oplus a_1c_1 \oplus b_1c_1 \oplus a_1b_1, & d_4 &= a_2c_2 \oplus b_2c_2 \oplus a_2b_2, \\ d_2 &= a_1c_2 \oplus b_1c_2, & d_5 &= a_1b_2, \\ d_3 &= a_2 \oplus a_2c_1 \oplus b_2c_1, & d_6 &= a_2b_1. \end{aligned} \tag{10}$$

*Construction of $\mathcal{R}$ and $\mathcal{C}$.* It is clear that if $n \times s_{in} < m \times s_{out}$, the output sharing can not be uniform. Even if $n \times s_{in} \geq m \times s_{out}$ the uniformity is not guaranteed. The output sharing described in Equations (8), (9) and (10) are such non-uniform examples which require refreshing ($\mathcal{R}$ layer). An alternative approach for the first-order case only is to decrease the number of shares in order to achieve uniformity after storing the output of the mentioned sharings in registers (prior to the $\mathcal{C}$ layer). To exemplify, consider the following sharing of $d = ab \oplus c$ with 2 input shares of each variable.

$$d_1 = a_1b_1 \oplus c_1, \qquad d_2 = a_1b_2, \qquad d_3 = a_2b_1 \oplus c_2, \qquad d_4 = a_2b_2. \tag{11}$$

The sharing $\mathbf{d}$ is a nonuniform 4-sharing. However, the 2-sharing $\mathbf{e}$ generated by $e_1 = d_1 \oplus d_2$ and $e_2 = d_3 \oplus d_4$ is uniform. Moreover, if the sharing $\mathbf{d}$ is stored in registers before decreasing the number of shares, as implied by the registers between the $\mathcal{R}$ and $\mathcal{C}$ layers in the generalized construction, any glitch during the calculation of $e_i$ does not reveal information about the input values. Note that the selection of the xored terms is not random at all and must be performed with extreme care. A bad choice for a compression layer would be $e_1 = d_1 \oplus d_3$ and $e_2 = d_2 \oplus d_4$, since $e_2 = (a_1 \oplus a_2)b_2 = ab_2$ obviously reveals information on $a$.

*Security argument of the improved bound on the number of shares.* It is noteworthy that the security of the improved scheme is not proven using indices as for the generalized scheme described in Section 3.4. Instead, since we assume that each input sharing is independent of the others, we ensure that any combination of $d$ probes miss at least one share of each input variable. Therefore $d+1$ input shares are sufficient in order to provide non-completeness in $\mathcal{N}$ hence independence of the output shares from each unmasked input. As discussed above the requirements that should be satisfied by the $\mathcal{L}$ and $\mathcal{C}$ layers in order to provide the claimed security highly depends on the function. Therefore, we avoid to give a generic construction besides imposing $d^{\text{th}}$-order non-completeness in $\mathcal{L}$ and extreme care not to unmask in $\mathcal{C}$. The extension to higher orders is straightforward under given exceptions.

*Application to 4-bit quadratic permutations.* In order to increase the usability of this technique, we provide a possible sharing $\mathbf{S}$ secure against 2-probing adversary for one permutation $S$ from each 4-bit quadratic affine equivalence class of permutation in Appendix A. Any other permutation $S'$ that is affine equivalent to $S$ can be calculated by $S' = A \circ S \circ B$ where $A$ and $B$ refer to affine transformations which can be shared as described in Section 2. Note that there exist 6 such quadratic classes which can be used to generate half of the 4-bit cubic permutations as described in [6]. This set of permutations covers a significant part of all $4 \times 4$ S-boxes.

*Connections with software ISW.* In [25], a fast masked AES at any order is given. The authors improve the security guarantees with respect to the number of shares from $s = 2d+1$ to $s = d+1$. This improvement actually resembles to the contribution of this section. The improvement was later shown to be slightly flawed by [10]. However, we observe here that the refreshing from [25] is not exactly the same as the layer $\mathcal{R}$ presented in Section 3.3 (operating in GF($2^8$)). Namely, the refreshing from [25] uses 1 unit of randomness (elements in GF($2^8$)) less than $\mathcal{R}$. Using a refreshing that mimics the layer $\mathcal{R}$ makes the second-order flaw disappear [2].

### 5.2 Resistance against distance leakage

There are many applications of the ISW scheme for masked software implementations [25], [13]. In the case of AES, there is a significant performance gain if the ISW operates in GF($2^8$), due to the algebraic structure of the AES Sbox. A common problem with ISW-based software implementations is the mismatch between the probing model in which ISW is proven secure and

the leakage behavior of the device that runs the implementation. For instance, typical processors can be approximately modeled by a distance-based leakage behavior (Hamming distance) rather than value-based one (Hamming weight). Thus, a straightforward implementation of ISW without a careful prior profiling of the device leakage behavior will likely lead to an insecure implementation. This is because, even if ISW is secure in weight-based leakage behavior, it is not in a distance-based one.

There are already some theoretical solutions for this problem, although they come with a significant cost [9], [1]. We point out here that it is possible to minimize the exposure to this issue with the same modification performed in Section 3, i.e. bringing the non-completeness condition.

The generalized scheme (e.g. after the second modification in Figure 7) computes sequentially each branch (component function) $F_i, i = 1, 2, 3$ and then performs a refreshing. This scheme is secure even if during the computation of each branch $F_i$ the device leaks distances (or a more complex leakage function of several values). Contrary to the ISW, we do not impose specific constraints on the order of evaluation of intermediates (within each $F_i$). This result immediately follows from non-completeness of each branch $F_i$. It is required, however, to make sure that there is no distance leakage between an intermediate appearing in $F_i$ and another in $F_j$, for $i \neq j$. It is noteworthy that the randomness requirement, running time and memory requirements stay the same as in the original algorithm.

## 6 Conclusion

In this paper, we explored the connections, similitudes and differences between several masking schemes, both from theoretical domains and from practitioners working under real-world constraints. It is remarkable how two substantially disparate communities arrive to essentially similar designs. This perhaps builds even more confidence on the underlying techniques.

There are certainly many future avenues of research. For example, it would be desirable to have explicit and tight bounds on the randomness requirements to achieve efficient masked implementations.

*Acknowledgements.* The authors would like to thank the CRYPTO 2015 reviewers for their valuable comments, as well as Fré Vercauteren and Vincent Rijmen for stimulating discussions. This work has been supported in part by the Research Council of KU Leuven (OT/13/071 and GOA/11/007), by the FWO G.0550.12N and by the Hercules foundation (AKUL/11/19). Oscar Reparaz is funded by a PhD fellowship of the Fund for Scientific Research - Flanders (FWO). Benedikt Gierlichs is a Postdoctoral Fellow of the Fund for Scientific Research - Flanders (FWO). Begül Bilgin is partially supported by the FWO project G0B4213N.

## References

1. J. Balasch, B. Gierlichs, V. Grosso, O. Reparaz, and F.-X. Standaert. On the Cost of Lazy Engineering for Masked Software Implementations. In *CARDIS*, volume 8968 of *LNCS*. Spinger, 2014.
2. G. Barthe, S. Belad, F. Dupressoir, P.-A. Fouque, B. Grégoire, and P.-Y. Strub. Verified proofs of higher-order masking. In *EUROCRYPT*, volume 9056 of *LNCS*, pages 457–485. Springer, 2015.
3. B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen. Higher-order threshold implementations. In *ASIACRYPT*, volume 8874 of *LNCS*, pages 326–343. Springer, 2014.
4. B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen. A more efficient AES threshold implementation. In *AFRICACRYPT*, volume 8469 of *LNCS*, pages 267–284. Springer, 2014.
5. B. Bilgin, S. Nikova, V. Nikov, V. Rijmen, and G. Stütz. Threshold implementations of all $3 \times 3$ and $4 \times 4$ S-boxes. In *CHES*, volume 7428 of *LNCS*, pages 76–91. Springer, 2012.

6. B. Bilgin, S. Nikova, V. Nikov, V. Rijmen, N. Tokareva, and V. Vitkup. Threshold implementations of small S-boxes. *Cryptography and Communications*, 7(1):3–33, 2015.
7. S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi. Towards sound approaches to counteract power-analysis attacks. In *CRYPTO*, volume 1666 of *LNCS*, pages 398–412. Springer, 1999.
8. J.-S. Coron. Higher order masking of look-up tables. In *EUROCRYPT*, volume 8441 of *LNCS*, pages 441–458. Springer, 2014.
9. J.-S. Coron, C. Giraud, E. Prouff, S. Renner, M. Rivain, and P. K. Vadnala. Conversion of security proofs from one leakage model to another: A new issue. In *COSADE*, volume 7275 of *LNCS*, pages 69–81. Springer, 2012.
10. J.-S. Coron, E. Prouff, M. Rivain, and T. Roche. Higher-order side channel security and mask refreshing. In *FSE*, volume 8424 of *LNCS*, pages 410–424. Springer, 2013.
11. A. Duc, S. Dziembowski, and S. Faust. Unifying leakage models: From probing attacks to noisy leakage. In *EUROCRYPT*, volume 8441 of *LNCS*, pages 423–440. Springer, 2014.
12. L. Goubin and J. Patarin. DES and differential power analysis the duplication method. In *CHES*, volume 1717 of *LNCS*, pages 158–172. Springer, 1999.
13. V. Grosso, G. Leurent, F.-X. Standaert, and K. Varici. LS-Designs: Bitslice Encryption for Efficient Masked Software Implementations. In *FSE*, LNCS. Springer, 2014.
14. Y. Ishai, A. Sahai, and D. Wagner. Private circuits: Securing hardware against probing attacks. In *CRYPTO*, volume 2729 of *LNCS*, pages 463–481. Springer, 2003.
15. P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *CRYPTO*, volume 1666 of *LNCS*, pages 388–397. Springer, 1999.
16. S. Kutzner, P. H. Nguyen, and A. Poschmann. Enabling 3-share threshold implementations for all 4-bit s-boxes. In *ICISC*, pages 91–108. Springer, 2013.
17. S. Mangard, T. Popp, and B. M. Gammel. Side-channel leakage of masked CMOS gates. In *CT-RSA*, volume 3376 of *LNCS*, pages 351–365. Springer, 2005.
18. S. Mangard and K. Schramm. Pinpointing the side-channel leakage of masked AES hardware implementations. In *CHES*, volume 4249 of *LNCS*, pages 76–90. Springer, 2006.
19. T. S. Messerges. Using second-order power analysis to attack dpa resistant software. In *CHES*, volume 1965 of *LNCS*, pages 238–251. Springer, 2000.
20. A. Moradi, A. Poschmann, S. Ling, C. Paar, and H. Wang. Pushing the limits: A very compact and a threshold implementation of AES. In *EUROCRYPT*, volume 6632 of *LNCS*, pages 69–88. Springer, 2011.
21. S. Nikova, C. Rechberger, and V. Rijmen. Threshold implementations against side-channel attacks and glitches. In *ICS*, volume 4307, pages 529–545. Springer, 2006.
22. A. Poschmann, A. Moradi, K. Khoo, C.-W. Lim, H. Wang, and S. Ling. Side-channel resistant crypto for less than 2,300 GE. *Journal of Cryptology*, 24(2):322–345, 2011.
23. O. Reparaz. A note on the security of higher-order threshold implementations. Cryptology ePrint Archive, Report 2015/001.
24. O. Reparaz, B. Gierlichs, and I. Verbauwhede. Selecting time samples for multivariate DPA attacks. In *CHES*, volume 7428 of *LNCS*, pages 155–174. Springer, 2012.
25. M. Rivain and E. Prouff. Provably secure higher-order masking of AES. In *CHES*, volume 6225 of *LNCS*, pages 413–427. Springer, 2010.
26. T. Schneider and A. Moradi. Leakage assessment methodology - a clear roadmap for side-channel evaluations. In *CHES*, LNCS. 2015.
27. K. Tiri, M. Akmal, and I. Verbauwhede. A dyamic and differential CMOS logic with signal independent power consumption to withstand differential power analysis on smart cards. In *ESSCIRC*, pages 403–406, 2002.
28. K. Tiri and I. Verbauwhede. A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation. In *DATE*, 2004.
29. E. Trichina. Combinational logic design for aes subbyte transformation on masked data. Cryptology ePrint Archive, Report 2003/236.

# A First-order Masking of Quadratic 4-bit Permutations with $d+1$ Shares

We use the notation in [6] in order to represent a class. Namely, $Q_i^j$ corresponds to a quadratic class of $j$ bits with the class number $i$ where the classes are order lexicographically from their representatives. Each permutation $S(a,b,c,d) = (x,y,z,t)$ has 4 input and output bits. The component functions $F, G, H, K$ outputs $x, y, z, t$ respectively. $x$ (resp. $a$) is the most significant bit whereas $t$ (resp. $d$) is the least significant bit. We only consider first-order security with 2 input shares. The sharing $\mathbf{x}$ of an output variable $x$ refers to its sharing after $N$ followed by $L_1$. $\mathbf{x}$ is in some cases not uniform and requires refreshing if used as is. We also describe the 2-sharing $\bar{\mathbf{x}}$ with shares $\bar{x}_i$ after $L_2$ layer such that the sharing is uniform.

## A.1 Class $Q_4^4$

$S = [0,1,2,3,4,5,6,7,8,9,10,11,13,12,15,14]$

$$
\begin{aligned}
x &= F(a,b,c,d) = a \\
y &= G(a,b,c,d) = b \\
z &= H(a,b,c,d) = c \\
t &= K(a,b,c,d) = ab \oplus d
\end{aligned}
$$

$$
\begin{aligned}
x_1 &= F_1(\mathbf{a,b,c,d}) = a_1 & & \\
x_2 &= F_2(\mathbf{a,b,c,d}) = a_2 & \bar{x}_1 &= x_1 \\
y_1 &= G_1(\mathbf{a,b,c,d}) = b_1 & \bar{x}_2 &= x_2 \\
y_2 &= G_2(\mathbf{a,b,c,d}) = b_2 & \bar{y}_1 &= y_1 \\
z_1 &= H_1(\mathbf{a,b,c,d}) = c_1 & \bar{y}_2 &= y_2 \\
z_2 &= H_2(\mathbf{a,b,c,d}) = c_2 & \bar{z}_1 &= z_1 \\
t_1 &= K_1(\mathbf{a,b,c,d}) = a_1 b_1 \oplus d_1 & \bar{z}_2 &= z_2 \\
t_2 &= K_2(\mathbf{a,b,c,d}) = a_1 b_2 & \bar{t}_1 &= t_1 \oplus t_2 \\
t_3 &= K_3(\mathbf{a,b,c,d}) = a_2 b_1 & \bar{t}_2 &= t_3 \oplus t_4 \\
t_4 &= K_4(\mathbf{a,b,c,d}) = a_2 b_2 \oplus d_2 & &
\end{aligned}
$$

## A.2 Class $Q_{12}^4$

$S = [0,1,2,3,4,5,6,7,8,9,12,13,14,15,10,11]$

$$
\begin{aligned}
x &= F(a,b,c,d) = a \\
y &= G(a,b,c,d) = ac \oplus b \\
z &= H(a,b,c,d) = ab \oplus ac \oplus c \\
t &= K(a,b,c,d) = d
\end{aligned}
$$

$$
\begin{aligned}
x_1 &= F_1(\mathbf{a,b,c,d}) = a_1 \\
x_2 &= F_2(\mathbf{a,b,c,d}) = a_2 \\
y_1 &= G_1(\mathbf{a,b,c,d}) = a_1c_1 \oplus b_1 & \bar{x}_1 &= x_1 \\
y_2 &= G_2(\mathbf{a,b,c,d}) = a_1c_2 & \bar{x}_2 &= x_2 \\
y_3 &= G_3(\mathbf{a,b,c,d}) = a_2c_1 \oplus b_2 & \bar{y}_1 &= y_1 \oplus y_2 \\
y_4 &= G_4(\mathbf{a,b,c,d}) = a_2c_2 & \bar{y}_2 &= y_3 \oplus y_4 \\
z_1 &= H_1(\mathbf{a,b,c,d}) = a_1b_1 \oplus a_1c_1 \oplus c_1 & \bar{z}_1 &= z_1 \oplus z_2 \\
z_2 &= H_2(\mathbf{a,b,c,d}) = a_1b_2 \oplus a_1c_2 & \bar{z}_2 &= z_3 \oplus z_4 \\
z_3 &= H_3(\mathbf{a,b,c,d}) = a_2b_1 \oplus a_2c_1 & \bar{t}_1 &= t_1 \\
z_4 &= H_4(\mathbf{a,b,c,d}) = a_2b_2 \oplus a_2c_2 \oplus c_2 & \bar{t}_2 &= t_2 \\
t_1 &= K_1(\mathbf{a,b,c,d}) = d_1 \\
t_2 &= K_2(\mathbf{a,b,c,d}) = d_2
\end{aligned}
$$

### A.3 Class $Q^4_{293}$

$S = [0, 1, 2, 3, 4, 5, 7, 6, 8, 9, 12, 13, 14, 15, 11, 10]$

$$
\begin{aligned}
x &= F(a, b, c, d) = a \\
y &= G(a, b, c, d) = ac \oplus b \\
z &= H(a, b, c, d) = ab \oplus ac \oplus c \\
t &= K(a, b, c, d) = bc \oplus d
\end{aligned}
$$

$$
\begin{aligned}
x_1 &= F_1(\mathbf{a,b,c,d}) = a_1 \\
x_2 &= F_2(\mathbf{a,b,c,d}) = a_2 \\
y_1 &= G_1(\mathbf{a,b,c,d}) = a_1c_1 \oplus b_1 \\
y_2 &= G_2(\mathbf{a,b,c,d}) = a_1c_2 & \bar{x}_1 &= x_1 \\
y_3 &= G_3(\mathbf{a,b,c,d}) = a_2c_1 \oplus b_2 & \bar{x}_2 &= x_2 \\
y_4 &= G_4(\mathbf{a,b,c,d}) = a_2c_2 & \bar{y}_1 &= y_1 \oplus y_2 \\
z_1 &= H_1(\mathbf{a,b,c,d}) = a_1b_1 \oplus a_1c_1 \oplus c_1 & \bar{y}_2 &= y_3 \oplus y_4 \\
z_2 &= H_2(\mathbf{a,b,c,d}) = a_1b_2 \oplus a_1c_2 & \bar{z}_1 &= z_1 \oplus z_2 \\
z_3 &= H_3(\mathbf{a,b,c,d}) = a_2b_1 \oplus a_2c_1 & \bar{z}_2 &= z_3 \oplus z_4 \\
z_4 &= H_4(\mathbf{a,b,c,d}) = a_2b_2 \oplus a_2c_2 \oplus c_2 & \bar{t}_1 &= t_1 \oplus t_2 \\
t_1 &= K_1(\mathbf{a,b,c,d}) = b_1c_1 \oplus d_1 & \bar{t}_2 &= t_3 \oplus t_4 \\
t_2 &= K_2(\mathbf{a,b,c,d}) = b_1c_2 \\
t_3 &= K_3(\mathbf{a,b,c,d}) = b_2c_1 \oplus d_2 \\
t_4 &= K_4(\mathbf{a,b,c,d}) = b_2c_2
\end{aligned}
$$

### A.4 Class $Q^4_{294}$

$S = [0, 1, 2, 3, 4, 5, 7, 6, 8, 9, 12, 13, 14, 15, 11, 10]$

$$
\begin{aligned}
x &= F(a,b,c,d) = a \\
y &= G(a,b,c,d) = b \\
z &= H(a,b,c,d) = ab \oplus c \\
t &= K(a,b,c,d) = ac \oplus d
\end{aligned}
$$

$$
\begin{aligned}
x_1 &= F_1(\mathbf{a,b,c,d}) = a_1 \\
x_2 &= F_2(\mathbf{a,b,c,d}) = a_2 \\
y_1 &= G_1(\mathbf{a,b,c,d}) = b_1 \\
y_2 &= G_2(\mathbf{a,b,c,d}) = b_2 \\
z_1 &= H_1(\mathbf{a,b,c,d}) = a_1 b_1 \oplus c_1 \\
z_2 &= H_2(\mathbf{a,b,c,d}) = a_1 b_2 \\
z_3 &= H_3(\mathbf{a,b,c,d}) = a_2 b_1 \oplus c_2 \\
z_4 &= H_4(\mathbf{a,b,c,d}) = a_2 b_2 \\
t_1 &= K_1(\mathbf{a,b,c,d}) = a_1 c_1 \oplus d_1 \\
t_2 &= K_2(\mathbf{a,b,c,d}) = a_1 c_2 \\
t_3 &= K_3(\mathbf{a,b,c,d}) = a_2 c_1 \oplus d_2 \\
t_4 &= K_4(\mathbf{a,b,c,d}) = a_2 c_2
\end{aligned}
\qquad
\begin{aligned}
\bar{x}_1 &= x_1 \\
\bar{x}_2 &= x_2 \\
\bar{y}_1 &= y_1 \\
\bar{y}_2 &= y_2 \\
\bar{z}_1 &= z_1 \oplus z_2 \\
\bar{z}_2 &= z_3 \oplus z_4 \\
\bar{t}_1 &= t_1 \oplus t_2 \\
\bar{t}_2 &= t_3 \oplus t_4
\end{aligned}
$$

### A.5 Class $Q^{4}_{299}$

$S = [0, 1, 2, 3, 4, 5, 7, 8, 10, 12, 14, 11, 9, 15, 13]$

$$
\begin{aligned}
x &= F(a,b,c,d) = a \\
y &= G(a,b,c,d) = ab \oplus ac \oplus b \\
z &= H(a,b,c,d) = ab \oplus ac \oplus ad \oplus c \\
t &= K(a,b,c,d) = ab \oplus ad \oplus d
\end{aligned}
$$

$$
\begin{aligned}
x_1 &= F_1(\mathbf{a,b,c,d}) = a_1 \\
x_2 &= F_2(\mathbf{a,b,c,d}) = a_2 \\
y_1 &= G_1(\mathbf{a,b,c,d}) = a_1b_1 \oplus a_1c_1 \oplus b_1 \\
y_2 &= G_2(\mathbf{a,b,c,d}) = a_1b_2 \oplus a_1c_2 \\
y_3 &= G_3(\mathbf{a,b,c,d}) = a_2b_1 \oplus a_2c_1 \\
y_4 &= G_4(\mathbf{a,b,c,d}) = a_2b_2 \oplus a_2c_2 \oplus b_2 \\
z_1 &= H_1(\mathbf{a,b,c,d}) = a_1b_1 \oplus a_1c_1 \oplus a_1d_1 \oplus c_1 \\
z_2 &= H_2(\mathbf{a,b,c,d}) = a_1b_2 \oplus a_1c_2 \oplus a_1d_2 \\
z_3 &= H_3(\mathbf{a,b,c,d}) = a_2b_1 \oplus a_2c_1 \oplus a_2d_1 \\
z_4 &= H_4(\mathbf{a,b,c,d}) = a_2b_2 \oplus a_2c_2 \oplus a_2c_2 \oplus c_2 \\
t_1 &= K_1(\mathbf{a,b,c,d}) = a_1b_1 \oplus a_1d_1 \oplus d_1 \\
t_2 &= K_2(\mathbf{a,b,c,d}) = a_1b_2 \oplus a_1d_2 \\
t_3 &= K_3(\mathbf{a,b,c,d}) = a_2b_1 \oplus a_2d_1 \\
t_4 &= K_4(\mathbf{a,b,c,d}) = a_2b_2 \oplus a_2d_2 \oplus d_2
\end{aligned}
\qquad
\begin{aligned}
\bar{x}_1 &= x_1 \\
\bar{x}_2 &= x_2 \\
\bar{y}_1 &= y_1 \oplus y_2 \\
\bar{y}_2 &= y_3 \oplus y_4 \\
\bar{z}_1 &= z_1 \oplus z_2 \\
\bar{z}_2 &= z_3 \oplus z_4 \\
\bar{t}_1 &= t_1 \oplus t_2 \\
\bar{t}_2 &= t_3 \oplus t_4
\end{aligned}
$$

### A.6 Class $Q^4_{300}$

$S = [0, 1, 2, 3, 4, 5, 8, 9, 13, 12, 7, 6, 11, 10, 15, 14]$

$$
\begin{aligned}
x &= F(a, b, c, d) = ac \oplus bc \oplus a \\
y &= G(a, b, c, d) = bc \oplus a \oplus b \\
z &= H(a, b, c, d) = ab \oplus bc \oplus c \\
t &= K(a, b, c, d) = a \oplus d
\end{aligned}
$$

$$
\begin{aligned}
x_1 &= F_1(\mathbf{a,b,c,d}) = a_1c_1 \oplus b_1c_1 \oplus a_1 \\
x_2 &= F_2(\mathbf{a,b,c,d}) = a_1c_2 \oplus b_1c_2 \\
x_3 &= F_3(\mathbf{a,b,c,d}) = a_2c_1 \oplus b_2c_1 \oplus a_2 \\
x_4 &= F_4(\mathbf{a,b,c,d}) = a_2c_2 \oplus b_2c_2 \\
y_1 &= G_1(\mathbf{a,b,c,d}) = b_1c_1 \oplus a_1 \oplus b_1 \\
y_2 &= G_2(\mathbf{a,b,c,d}) = b_1c_2 \\
y_3 &= G_3(\mathbf{a,b,c,d}) = b_2c_1 \\
y_4 &= G_4(\mathbf{a,b,c,d}) = b_2c_2 \oplus a_2 \oplus b_2 \\
z_1 &= H_1(\mathbf{a,b,c,d}) = a_1b_1 \oplus b_1c_1 \oplus c_1 \\
z_2 &= H_2(\mathbf{a,b,c,d}) = a_1b_2 \\
z_3 &= H_3(\mathbf{a,b,c,d}) = b_2c_1 \\
z_4 &= H_4(\mathbf{a,b,c,d}) = a_2b_1 \\
z_5 &= H_5(\mathbf{a,b,c,d}) = b_2c_1 \\
z_6 &= H_6(\mathbf{a,b,c,d}) = a_2b_2 \oplus b_2c_2 \oplus c_2 \\
t_1 &= K_1(\mathbf{a,b,c,d}) = a_1 \oplus d_1 \\
t_2 &= K_2(\mathbf{a,b,c,d}) = a_2 \oplus d_2
\end{aligned}
\qquad
\begin{aligned}
\bar{x}_1 &= x_1 \oplus x_2 \\
\bar{x}_2 &= x_3 \oplus x_4 \\
\bar{y}_1 &= y_1 \oplus y_2 \\
\bar{y}_2 &= y_3 \oplus y_4 \\
\bar{z}_1 &= z_1 \oplus z_2 \oplus z_3 \\
\bar{z}_2 &= z_4 \oplus z_5 \oplus z_6 \\
\bar{t}_1 &= t_1 \\
\bar{t}_2 &= t_2
\end{aligned}
$$