Automated Generation of Masked Nonlinear Components: From Lookup Tables to Private Circuits

Lixuan Wu[1,2], Yanhong Fan[1,2,3], Bart Preneel[4], Weijia Wang[1,2,3] and Meiqin Wang[1,2,3](✉)

[1] School of Cyber Science and Technology, Shandong University, Qingdao, China
{yanhongfan,mqwang,weijiawang}@sdu.edu.cn,lixuanwu@mail.sdu.edu.cn
[2] Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education, Shandong University, Jinan, China
[3] Quan Cheng Shandong Laboratory, Jinan, China
[4] imec-COSIC, KU Leuven, Belgium
bart.preneel@esat.kuleuven.be

Abstract. Masking is considered to be an essential defense mechanism against side-channel attacks, but it is challenging to be adopted for hardware cryptographic implementations, especially for high-security orders. Recently, Knichel et al. proposed an automated tool called AGEMA that enables the generation of masked implementations in hardware for arbitrary security orders using composable gadgets. This accelerates the construction and practical application of masking schemes. This article proposes a new automated tool named AGMNC that can generate masked nonlinear components with much better performance. The effectiveness of AGMNC is evaluated in several case studies. The evaluation results show a significant performance improvement, particularly for the first-order secure SKINNY S-box: saving 41% area, 25% latency, and 49% dynamic power. We achieve such a good result by integrating three key techniques: a new composable AND-XOR gadget, an optimization strategy based on the latency asymmetry feature of the AND-XOR gadget, and an implementation optimization for synchronization. Besides, we use the formal verification tool SILVER and FPGA-based practical experiments to confirm the security of the masked implementations generated by AGMNC.

Keywords: Side-Channel Analysis · Masking · Composable Gadget · AGMNC

1 Introduction

With the rapid growth of the Internet of Things (IoT), the number of connected devices has increased significantly. IoT devices are attractive targets for a range of attacks; in particular, the easy access to these devices renders them vulnerable to physical attacks. Among these physical attacks, Side-Channel Analysis (SCA) attacks [Koc96, KJJ99] have gained significant attention from researchers and practitioners due to their ability to extract secret information from the devices without the need for direct access to the internal components. SCA attacks can exploit various physical properties, such as timing [Koc96], power consumption [KJJ99], electromagnetic (EM) emanations [GMO01], temperature and heat dissipation [HS14], to extract secret information processed by the device. In response to the severe threat posed by SCA attacks, numerous approaches have been proposed to mitigate this risk. Among approaches, masking [CJRR99] has emerged as the most widely studied and deployed countermeasure due to its sound theoretical foundations. For example, the first necessary requirement of a masking scheme is the Ishai-Sahai-Wagner (ISW) d-probing model [ISW03] that ensures that any d internal variables are independently distributed from the secret input.

However, it is still non-trivial to adopt masking in practice, since d-probing security is invalid in the presence of many known physical defaults. For instance, many masking schemes (see, e.g., [ISW03, GMK17, GM18] for an incomplete list) were shown to be insecure in hardware. It is mainly because they fall short in resisting glitches that are known to be the most challenging hardware physical default to overcome. In this respect, the glitch-extended probing model [FGP$^+$18] has been proposed to formalize glitches.

Although introducing some simple, practical, and formal adversary models has facilitated the design and security verification of masking schemes, designing masking schemes with high-security orders for complex circuits remains challenging due to the high computational cost. Following a divide-and-conquer strategy, researchers have defined some composable security notions that allow large circuits to be constructed by sub-circuits satisfying composability, also known as gadgets. In this way, constructing large circuits is reduced to the construction of small ones satisfying composability. Those security notions include NI [BBD$^+$15], SNI [BBD$^+$16], PINI [CS20], and so on. Also, the glitch-extended probing model and composable security notions are also combined. To further promote the practical application of composability, an automation tool called AGEMA was introduced by Knichel et al. [KMMS21], which allows designers to generate hardware masked implementation from an unprotected implementation using composable gadgets.

We note that, albeit AGEMA can easily generate masked hardware circuit from a simple but unprotected design, there still exists a large hardware performance gap between masked circuits generated from AGEMA and manually designed ones. In this paper, based on the fact that nonlinear components of the whole circuit are the most complex and difficult part of masking, we propose a software tool AGMNC that takes some architecture-level optimizations into account to reduce the gap.

1.1 Contributions

AGMNC can automatically generate hardware masked circuits from the look-up table description of nonlinear components, such as an S-box. To generate a more efficient hardware masked implementation, we proposes a new composable gadget (i.e., an AND-XOR gadget) and two key optimization techniques (i.e., latency asymmetry and implementation optimization).

A new composable AND-XOR gadget. There are usually XOR operations between the quadratic terms and linear terms in the Boolean expressions of an S-box. Consider, for example, the Boolean function $f = ab + c$: this function is realized in AGEMA by trivially combining an HPC-AND gadget and XOR operations. The implementation in AGEMA requires the insertion of additional registers in the input path of each share of primary input c to synchronize the output shares. To overcome this disadvantage, we propose an AND-XOR gadget, which considers the HPC-AND gadget and XOR operations jointly. Compared to the trivial combination, the AND-XOR gadget saves $d + 1$ registers, where d is the security order: a circuit is secure against attacks or order d if it can resist an attacker that combines d measurements for each trace.

Two optimizations. In addition to the AND-XOR gadget, AGMNC integrates two key techniques to improve the hardware performance of the masked implementation, namely latency asymmetry and implementation optimization. The latency asymmetry means that the latency from each input port to the output port is different in a gadget. The HPC-AND gadget, for example, has a latency from the input port to the output port of 1 cycle and 2 cycles, respectively. By using the properties of AND-XOR and HPC-AND gadgets, this technique significantly reduces the latency and area of the final implementation. Based on the observation and analysis, we describe a new optimization technique to synchronize the latency of the final implementation. This technique requires fewer registers than synchronization without optimization.

An automation tool AGMNC. Based on the previous gadget and key techniques, we have developed a new automation tool AGMNC. The tool takes a look-up table as input and automatically generates the masked circuit. To illustrate the effectiveness of this tool, we apply AGMNC to several S-boxes. The results show a significant improvement in the hardware performance of the masked implementations generated by AGMNC. More specifically, for the first-order secure SKINNY S-box, AGMNC achieves a maximum reduction of 41%, 25%, and 49% in area, latency, and dynamic power, respectively, compared to AGEMA. Further, we apply the formal verification tool SILVER [KSM20] and FPGA-based practical experiments to confirm the security of the masked implementations.

1.2 Outline

We first present some necessary notions in Sect. 2. In Sect. 3, we highlight a new gadget and two key techniques applied in AGMNC, including the AND-XOR gadget, the latency asymmetry feature and the implementation optimization. To further illustrate the efficiency of AGMNC, Sect. 4 instantiates several S-boxes and full ciphers as case studies. In Sect. 5, we offer theoretical and experimental security analysis for the final masked implementations generated by AGMNC. We conclude our work in Sect. 6.

2 Preliminaries

In this section, we introduce the concepts and preliminary knowledge helpful to understand the rest of the article: Boolean masking, probing security, composable masking schemes, hardware private circuits, and the automation tool AGEMA.

2.1 Boolean Masking

We denote a binary random variable with lower-case italic x, the i-th share of a variable with x_i. A capital $X(\in \mathbb{F}_2^n, n > 1)$ represents a binary random vector, while X_j denotes the j-th shares of a vector X.

Boolean masking based on secret sharing has gained significant attention in hardware security as an essential defense against SCA attacks. The Boolean masking of a secret vector $X \in \mathbb{F}_2^n$ consists of s independent and random shares, denoted as $(X_0, X_1, \cdots, X_{s-1})$. It is necessary to ensure correctness by satisfying the condition $X = \bigoplus_{i=0}^{s-1} X_i$. Usually, the process of obtaining the above s shares involves two steps. Firstly, the X_i $(0 \le i \le s-2)$ are initialized with uniformly random strings. Secondly, X_{s-1} is derived as $X_{s-1} = (\bigoplus_{i=0}^{s-2} X_i) \oplus X$. Rather than performing leaking computations on the vector X, computations are performed on the shares X_i.

2.2 Probing Security

There are various models available to characterize and evaluate the security of masking schemes. Among them, the d-probing model [ISW03] has gained significant popularity and is widely used. In this model, the number d of probes reflects the order of the attack [BDF$^+$17, DDF14]. Since the d-probing model cannot characterize physical effects in hardware implementations, such as glitches, the model is limited to software implementations. Specifically, glitches are switching activities of wires in a circuit due to different delays of signals contributing to their intended values. To account for the impact of glitches, Faust et al. [FGP$^+$18] adapted the d-probing model and introduced the glitch-extended probing model. This model assumes that each probe placed on a combinatorial circuit propagates backward to the last synchronization point (e.g., registers). Since this article is related to hardware implementations, our evaluations and assessments are conducted under the glitch-extended probing model.

2.3 Composable Masking Schemes

Masking scheme designs that can achieve high order security remains a highly challenging task, even for an experienced designer. This encourages the development of composable gadgets, which are considered to be an efficient approach to designing masking schemes for complex functions. Specifically, composable gadgets are modules that realize atomic logic operations with specific properties under the glitch-extended probing model. Since these gadgets achieve particular properties, combining these gadgets to construct masking schemes for large circuits is possible. As a result, this approach of using composable gadgets simplifies the construction of masking schemes for complex circuits, as the focus is on finding gadgets realizing logic operations with specific properties rather than dealing with the whole complex circuits.

To achieve the composability of secure gadgets, Barthe et al. proposed the concept of Strong Non-Interferene (SNI) [BBD+16], which corrected the deficiencies of NI [BBD+15]. Under the concept of SNI, each probe placed on the output of the SNI-secure gadget is restricted to be perfectly simulatable without any information captured by this probe. Although SNI satisfies the composability of gadgets, it will lead to a large overhead with respect to fresh entropy and circuit area, especially for high-security orders. As a more efficient solution than SNI, Probe-Isolating Non-Interference (PINI) was introduced by Cassiers et al. in [CS20]. Based on the concept of share domain [GMK16], any probe was restricted to only propagate within its own share domain under the PINI. Formally, the concept of PINI can be described through Definition 1.

Definition 1 (d-Probe-Isolating Non-Interference)**.** Given a gadget G with secret input X, $X \in \mathbb{F}_2^n$, let t_i denotes probes placed on internal wires of G and t_o denotes probes placed on output wires of G, suth that $t_i + t_o \leq d$. The gadget G is d-PINI if and only if for all possible t_i and t_o, there exists a set of primary input indexes PI_i, with $|PI_i \leq t_i|$, primary output indexes PI_o, with $|PI_o \leq t_o|$, such that the observations of t_i and t_o can be perfectly simulated by $X_{PI_i \cup PI_o}$.

2.4 Hardware Private Circuits

Since PINI enables the trivial composition of hardware gadgets under the glitch-extended probing model, several concrete implementations of composable gadgets have been proposed. The HPC1 gadget introduced in [CGLS20] realizes the function of a 2-input AND gate and can be simply extended to arbitrary security orders. Specifically, HPC1 consists of a DOM-AND and a refresh gadget, where the sharing of one input of DOM-AND is refreshed through the refresh gadget. The number of fresh masks required by HPC1 is $d(d+1)/2 + [1, 2, 4, 5, 7, 9, 11, 12, 15, 17]$ [1] for security order $d \leq 10$, where the former is needed by DOM-AND and the latter is needed by the refresh gadget. Further, the second composable gadget HPC2 was introduced in the same work. The HPC2 is another construction for a 2-input AND gate that can be extended to arbitrary security orders. Compared to HPC1, HPC2 requires less fresh randomness, i.e., $d(d+1)/2$. Both HPC1 and HPC2 exhibit latency asymmetry feature: this means that if the first input sharing enters the HPC1 or HPC2 gadget at cycle k and another input sharing enters the gadget at cycle $k+1$, then the output can be generated at cycle $k+2$. Since each share-wise implementation of a linear function already satisfies the concept of PINI, the XOR operation or NOT operation can be realized without fresh randomness and without additional latency.

[1]This is a compact notation that indicates that for security order $d = 1$, 1 additional mask is required; for security order $d = 2$, 2 additional masks are required; for security order $d = 3$, 4 additional masks are required, and so on.

2.5 AGEMA

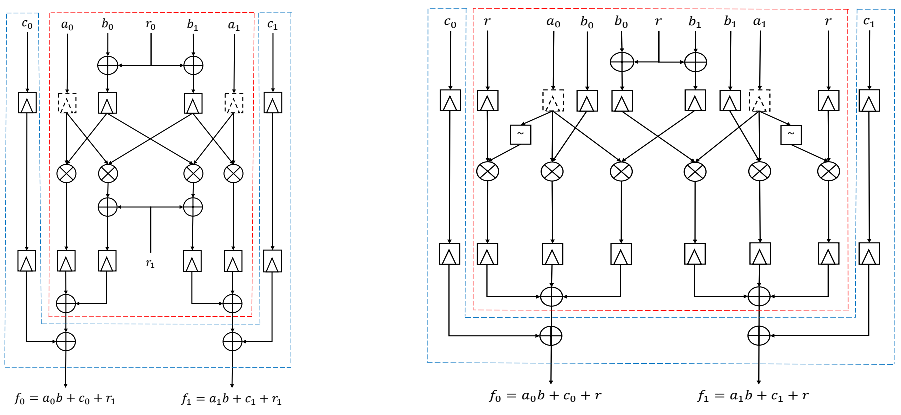
Knichel et al. [KMMS21] proposed an open-source software tool AGEMA, which makes it easy for designers to generate hardware masked circuits based on the unprotected HDL implementations. Based on the PINI concept, AGEMA supports several composable gadgets, including HPC1 and HPC2. Pipelining and clock gating are two synchronization techniques applied in AGEMA. Pipelining inserts additional registers to synchronize the input signals of each composable gadget, and clock gating modulates the clock signals of registers to achieve the same goal. Although pipelining requires a larger area overhead, it achieves better throughput than clock gating. Pipelining is more efficient than clock gating when large amounts of information are processed. Note that to provide a fair comparison, the hardware performance below related to AGEMA is generated using the pipelining synchronization technique.

3 New Techniques

In this section, we explain how AGMNC can generate more efficient masked implementations of S-boxes than AGEMA. The key techniques of AGMNC include a new AND-XOR gadget, latency asymmetry of the AND-XOR gadget, and implementation optimization.

3.1 AND-XOR Gadget

Cassiers et al. [CGLS20] introduced HPC1 and HPC2 to realize 2-input composable AND gadgets under the PINI notion in the glitch-extended probing model. As they can achieve arbitrary security orders, HPC1 and HPC2 are essential gadgets to be applied in AGEMA to generate masked implementations for security order $d \geq 1$. To facilitate the explanation and analysis, we utilize the term HPC-AND gadget to generally represent the implementation of a 2-input AND gadget using HPC1 or HPC2.

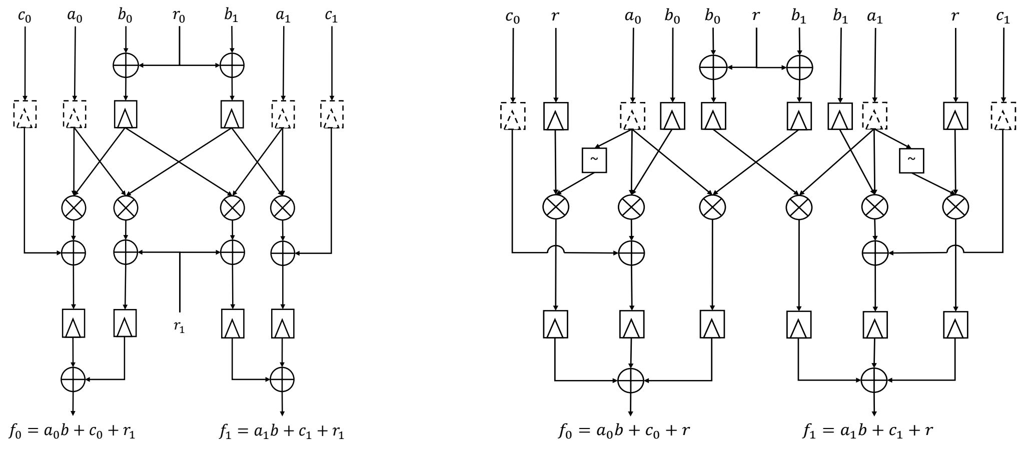


(a) First-order AND-XOR based on HPC1. (b) First-order AND-XOR based on HPC2.

Figure 1: AND-XOR in AGEMA.

There are usually XOR operations between the quadratic terms and linear terms in the Boolean expressions of nonlinear components, such as an S-box. The above scenario can be denoted as the Boolean function $f = ab + c$, where $a \in \mathbb{F}_2, b \in \mathbb{F}_2, c \in \mathbb{F}_2$. In AGEMA, the Boolean function $f = ab + c$ is realized using the HPC-AND gadget and XOR operations. The implementation for the first order security is depicted in Figs. 1(a) and 1(b), where the red dashed line is the HPC-AND gadget, and the blue dashed line is the XOR operations. Since the latency of a single HPC-AND is 2 cycles, it is necessary to insert two layers of registers in the path of the shares of primary input c (i.e., c_0, c_1) to synchronize the latency before performing the XOR operations.

After carefully analyzing the requirements, we construct two new compact designs for the two cases in Figs. 1(a) and 1(b). As an example, we provide a schematic overview of our designs for the first security order in Figs. 2(a) and 2(b), respectively. Although the dashed line registers (referred to as $Reg_{pipe}[]$ in Algorithms 1 and 2) are essential for synchronization and a pipelined architecture, they do not impact the security under the glitch-extended probing model. Both designs integrate the above HPC-AND gadget and XOR operations into a new gadget (called AND-XOR1 and AND-XOR2 gadget, respectively). Our new gadgets can achieve PINI security under the glitch-extended probing model and are generic for arbitrary security orders. From Figure 2, it can be seen that our new gadgets save a layer of registers in the path of the shares of primary input c. Since the c consists of at least $d+1$ shares, our designs generally reduce the number of registers by $d+1$ than the implementations in AGEMA, where d is the security order. Next, we present the construction principle and security analysis of our new gadgets.



(a) First-order AND-XOR1 gadget. (b) First-order AND-XOR2 gadget.

Figure 2: AND-XOR gadget.

Construction Principle. Algorithm 1 describes the generic algorithm-level of the AND-XOR1 gadget: the first order case is depicted in the Fig. 2(a). At the start of AND-XOR1 gadget (outlined in lines 1 to 11), a refresh gadget is essential to provide the desired security under the glitch-extended probing model. The details of the refresh gadget regarding the combinations of the shares of primary input b with some randomness are only provided for the cases of the first security order (outlined in lines 2 to 3) and the second security order (outlined in lines 5 to 8). The cases of security order $d \geq 3$ are given in the appendix of [CGLS20]. Then, some randomness is generated in lines 12 to 16. Lines 17 to 21 describe the cross-domain multiplications, i.e., the multiplications of two signals from different domains, the results of which are XOR-ed with randomness and then stored in registers. Lines 22 to 24 delineate three functionalities. Firstly, it multiplies signals that belong to the same domain. Subsequently, the results of the multiplications above are combined with the shares of primary input c utilizing XOR operations. Finally, the XOR operations are utilized once more in combination with the results of the cross-domain multiplications (lines 17 to 21). Note that in Algorithm 1, r_0, r_1, r_2 in the refresh gadget and r_{ij} in the cross-multiplications represent some randomness bits that are independent of each other. From Algorithm 1, it can be seen that the difference between various security orders of the AND-XOR1 gadget is the implementation of the refresh gadget (outlined in lines 1 to 11).

Algorithm 1 AND-XOR1 gadget

Input: shares $(a_i)_{0\leq i\leq d}$, $(b_i)_{0\leq i\leq d}$ and $(c_i)_{0\leq i\leq d}$, such that $\bigoplus_{i=0}^{d} a_i = a$, $\bigoplus_{i=0}^{d} b_i = b$ and $\bigoplus_{i=0}^{d} c_i = c$.

Output: shares $(f_i)_{0\leq i\leq d}$, such that $\bigoplus_{i=0}^{d} f_i = ab + c$.

```
1:  if $d = 1$ then
2:      $M[b_0] = Reg[b_0 \oplus r_0]$
3:      $M[b_1] = Reg[b_1 \oplus r_0]$
4:  else if $d = 2$ then
5:      $r_2 = Reg[r_0 \oplus r_1]$
6:      $M[b_0] = Reg[b_0 \oplus r_0]$
7:      $M[b_1] = Reg[b_1 \oplus r_1]$
8:      $M[b_2] = Reg[b_2 \oplus r_2]$
9:  else if $d \geq 3$ then
10:     refer to the appendix of [CGLS20].
11: end if
12: for $i = 0$ to $d$ do
13:     for $j = i + 1$ to $d$ do
14:         $r_{ij} = r_{ji}$, denotes a random bit.
15:     end for
16: end for
17: for $i = 0$ to $d$ do
18:     for $j = 0$ to $d$, $j \neq i$ do
19:         $u_{ij} = Reg[Reg_{pipe}[a_i] \otimes M[b_j] \oplus r_{ij}]$
20:     end for
21: end for
22: for $i = 0$ to $d$ do
23:     $f_i = Reg[Reg_{pipe}[a_i] \otimes M[b_i] \oplus Reg_{pipe}[c_i]] \oplus \bigoplus_{j=0,j\neq i}^{d} u_{ij}$
24: end for
```

Algorithm 2 AND-XOR2 gadget

Input: shares $(a_i)_{0\leq i\leq d}$, $(b_i)_{0\leq i\leq d}$ and $(c_i)_{0\leq i\leq d}$, such that $\bigoplus_{i=0}^{d} a_i = a$, $\bigoplus_{i=0}^{d} b_i = b$ and $\bigoplus_{i=0}^{d} c_i = c$.

Output: shares $(f_i)_{0\leq i\leq d}$, such that $f = \bigoplus_{i=0}^{d} f_i = ab + c$.

```
1:  for $i = 0$ to $d$ do
2:      for $j = i + 1$ to $d$ do
3:          $r_{ij} = r_{ji}$, denotes a random bit.
4:      end for
5:  end for
6:  for $i = 0$ to $d$ do
7:      for $j = 0$ to $d$, $j \neq i$ do
8:          $u_{ij} = Reg[Reg_{pipe}[a_i] \otimes Reg[r_{ij}]]$
9:          $v_{ij} = Reg[b_j \oplus r_{ij}]$
10:         $q_{ij} = Reg[Reg_{pipe}[a_i] \otimes v_{ij}]$
11:         $t_{ij} = u_{ij} \oplus q_{ij}$
12:     end for
13: end for
14: for $i = 0$ to $d$ do
15:     $f_i = Reg[Reg_{pipe}[a_i] \otimes Reg[b_i] \oplus Reg_{pipe}[c_i]] \oplus \bigoplus_{j=0,j\neq i}^{d}(t_{ij})$
16: end for
```

The AND-XOR2 gadget is our other design, shown in Algorithm 2. In addition, the first-order secure AND-XOR2 gadget is illustrated in Fig. 2(b). The first five lines of Algorithm 2 generate some randomness that will be used in the following operations. From lines 6 to 13, four signals, namely u_{ij}, v_{ij}, q_{ij} and t_{ij}, are defined. The signal u_{ij} is used to represent the results of $\overline{a_i} \otimes r_{ij}$. The signal v_{ij} masks the shares of the primary input b with randomness r_{ij} using XOR operations. The multiplications of two signals from different domains are computed by q_{ij}. The t_{ij} represents the combination of the results of $\overline{a_i} \otimes r_{ij}$ with the cross-domain multiplications (i.e., q_{ij}) using XOR operations. Lines 14 to 16 correspond to two parts, one is the multiplications of two signals from the same domain and the combination of the above multiplications with the shares of primary input c using XOR operations, and the other one is the combination of the cross-domain multiplications q_{ij} with u_{ij}. These final output shares of the AND-XOR2 gadget are generated by XORing the results of these two parts.

Security Analysis. Below, we provide Theorems 1 and 2 to prove the PINI security of AND-XOR1 and AND-XOR2 gadgets under the glitch-extended probing model.

Theorem 1. *Assuming that all randomness bits used in the AND-XOR1 gadget are statistically independent of each share of the primary inputs a, b and c, then the AND-XOR1 gadget is correct and PINI under the glitch-extended probing model.*

Proof. Correctness: According to the description of refresh gadget in [CGLS20], there is the following relation among the output shares of refresh gadget:

$$\bigoplus_{i=0}^{d} M[b_i] = \bigoplus_{i=0}^{d} b_i.$$

We skip all registers in Algorithm 1. Since $r_{ij} = r_{ji}$, it holds that

$$\begin{aligned}
f &= \bigoplus_{i=0}^{d} f_i \\
&= \bigoplus_{i=0}^{d} \left(a_i \otimes M[b_i] \oplus c_i \oplus \bigoplus_{j=0, j\neq i}^{d} (a_i \otimes M[b_j] \oplus r_{ij}) \right) \\
&= \bigoplus_{i=0}^{d} \left(a_i \otimes \bigoplus_{j=0}^{d} M[b_j] \oplus c_i \right) \oplus \bigoplus_{i=0}^{d} \left(\bigoplus_{j=0, j\neq i}^{d} (r_{ij}) \right) \\
&= \bigoplus_{i=0}^{d} \left(\bigoplus_{j=0}^{d} (a_i \otimes b_j) \oplus c_i \right) \\
&= \bigoplus_{i=0}^{d} \left(\bigoplus_{j=0}^{d} (a_i \otimes b_j) \right) \oplus \bigoplus_{i=0}^{d} c_i = a \otimes b \oplus c.
\end{aligned}$$

PINI: Considering each case of probe placement and arguing for simulatability, we prove that the AND-XOR1 gadget is PINI under the glitch-extended probing model.

i. When a probe placed on the input to $M[b_i], 0 \leq i \leq d$, we can observe the variables b_i and its responding 1-bit randomness r_i. This case can be denoted as $P_{b_i} = [b_i, r_i]$, which can be perfectly simulated by b_i and 1-bit fresh randomness r_i.

ii. When a probe placed on the input to u_{ij}, this case can be represented as $P_{u_{ij}} = [a_i, b_j \oplus r_j, r_{ij}]$, which can be perfectly simulated by a_i and randomness r_j and r_{ij}. If the additional variable u_{ji} is probed, this can be simulated by adding a_j to the simulation set. This is because the randomness r_i and r_j (used to mask b_i and b_j, respectively) are independent of each other. All other probes can be categorized into the above two cases, one with a single probe (i.e., u_{ij}) and the other one with a pair of probes whose subscripts are rotated (i.e., u_{ij} and u_{ji}). This is in line with the concept of PINI.

iii. The probe on f_i, i.e., $P_{f_i} = [a_i, b_i \oplus r_i, c_i] \cup \{\bigcup_{j=0,j\neq i}^{d} u_{ij}\}$ can be simulated by a_i, c_i and $b_i \oplus r_i$, which can be seen as a new random bit. If additionally P_{f_j} needs to be simulated, this can be done by following the description in step ii, i.e., adding the share a_j and c_j to the simulation set. This is because two output shares have at most one common cross-domain. This is in line with the PINI notion.

□

Theorem 2. *Assuming that all randomness bits used in the AND-XOR2 gadget are statistically independent of each share of the primary inputs a, b and c, then the AND-XOR2 gadget is correct and glitch-robust PINI.*

Proof. Correctness: We skip all registers in Algorithm 2. Since

$$\begin{aligned} t_{ij} &= u_{ij} \oplus q_{ij} \\ &= \overline{a_i} \otimes r_{ij} \oplus a_i \otimes (b_j \oplus r_{ij}) \\ &= a_i \otimes b_j \oplus r_{ij} \end{aligned}$$

and $r_{ij} = r_{ji}$, it holds that

$$\begin{aligned} f &= \bigoplus_{i=0}^{d} f_i \\ &= \bigoplus_{i=0}^{d} \left(a_i \otimes b_i \oplus c_i \oplus \bigoplus_{j=0,j\neq i}^{d} (a_i \otimes b_j \oplus r_{ij}) \right) \\ &= \bigoplus_{i=0}^{d} \left(\bigoplus_{j=0}^{d} (a_i \otimes b_j) \oplus c_i \right) \oplus \bigoplus_{i=0}^{d} \left(\bigoplus_{j=0,j\neq i}^{d} (r_{ij}) \right) \\ &= \bigoplus_{i=0}^{d} \left(\bigoplus_{j=0}^{d} (a_i \otimes b_j) \oplus c_i \right) \\ &= \bigoplus_{i=0}^{d} \left(\bigoplus_{j=0}^{d} (a_i \otimes b_j) \right) \oplus \bigoplus_{i=0}^{d} c_i = a \otimes b \oplus c. \end{aligned}$$

PINI: Arguing for each probe placement and simulatability case, we prove that the AND-XOR2 gadget is PINI under the glitch-extended probing model.

i. When a probe placed on the input to u_{ij}, we can observe the variables a_i and r_{ij}. It can be represented as $P_{u_{ij}} = [a_i, r_{ij}]$, which can be perfectly simulated by a_i and randomness r_{ij}.

ii. When a probe placed on the input to v_{ij}, this case can be represented as $P_{v_{ij}} = [b_j, r_{ij}]$, which can be perfectly simulated by b_j and randomness r_{ij}.

iii. The probe on q_{ij}, i.e., $P_{q_{ij}} = [a_i, b_j \oplus r_{ij}]$ an be simulated by $[a_i, r_{ij}]$. When an additional variable q_{ji} is probed, this can be achieved by adding a_j, b_i and b_j to the simulation set. All other probes can be done based on the above two cases, one is a single probe (i.e., q_{ij}) and the other one is a pair of probes whose subscripts are rotated (i.e., q_{ij} and q_{ji}).

iv. The probe on t_{ij}, i.e., $P_{t_{ij}} = [\overline{a_i} \otimes r_{ij}, a_i \otimes (b_j \oplus r_{ij})]$ can be simulated by a_i, r_{ij} and $b_j \oplus r_{ij}$, which can be seen as a new random bit. If additionally $P_{t_{ji}}$ needs to be simulated, this can be done by adding a_j, b_i, and b_j to the simulation set. The above case of considering two probes is in line with the notion of PINI. All other probes can be processed using the above two cases, one with a single probe (i.e., t_{ij}) and the other one with a pair of probes whose subscripts are rotated (i.e., t_{ij} and t_{ji}).

v. The probe on the output share, i.e., $P_{f_i} = [a_i, b_i, c_i] \cup \{\bigcup_{j=0, j\neq i}^{d} t_{ij}\}$ can be simulated by a_i, b_i, c_i and processing d times the random bit r_{ij} to simulate t_{ij}. From the expression of f_i, it can be seen that two output shares have at most one cross-domain in common, and using the same argument as described in step iv, i.e., adding some input shares from only one other domain and some additional randomness to the simulation set. This is in line with the notion of PINI.

□

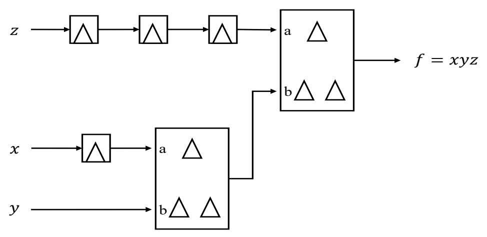
Compared with the implementations in AGEMA (as shown in Figs. 1(a) and 1(b)), the two new gadgets we proposed have $d+1$ fewer registers while maintaining the same requirements with respect to randomness and latency, where d is the security order. More specifically, a single AND-XOR1 gadget requires $d(d+1)/2 + [1, 2, 4, 5, 7, 9, 11, 12, 15, 17]$ bits of randomness and 2 cycles of latency. Similarly, a single AND-XOR2 gadget requires $d(d+1)/2$ bits of randomness and 2 cycles of latency. Due to the fewer registers, our new gadgets require a lower area than the implementations in AGEMA for the Boolean function $f = ab + c$. It is worth mentioning that, in addition to the area advantages of the Boolean function mentioned above, we believe that these two new gadgets can yield comparable results as two standalone gadgets in the S-box implementations. To facilitate the analysis of the common features of these two new gadgets, we utilize the term AND-XOR gadget to represent both the AND-XOR1 and AND-XOR2 gadgets generally in the remainder of this article.

3.2 Latency Asymmetry of AND-XOR Gadget

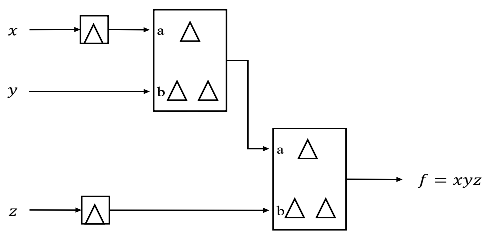
To further enhance the latency of the AND-XOR gadget, we conduct a detailed analysis of its latency asymmetry and integrate this feature into the automation tool AGMNC.

Latency asymmetry is a peculiar feature supported by the HPC-AND gadget. This feature arises because only the shares from one primary input of the HPC-AND gadget need to be refreshed. In other words, the HPC-AND gadget has two sorts of shares from primary input, one of which has 1 cycle of latency and the other has 2 cycles of latency. Interestingly, the AND-XOR gadget also exhibits this feature for the following reason: as shown in Algorithms 1 and 2, only the shares from primary input b need to be refreshed, while the shares from primary input a and c use $Reg_{pipe}[]$ to compensate for the latency asymmetry caused by the refresh of the shares from b. In other words, the AND-XOR gadget has three sorts of input ports, one with a latency of 2 cycles (marked as "b") and the remaining two with a latency of 1 cycle (marked as "a" and "c"). In the following, we demonstrate the impact of the latency asymmetric feature with several concrete examples.

To demonstrate the impact of utilizing the latency asymmetry, Fig. 3 presents two implementations with the Boolean function $f = xyz$ as an example. Note that Fig. 3 is a general schematic, where each signal is actually composed of $d+1$ shares and d is the security order of the masked implementation. Fig. 3(a) emulates the AGEMA implementation without considering the latency asymmetry, which requires $4(d+1)$ additional registers to synchronize the latency and yields an implementation with a latency of 4 cycles. On the other hand, Fig. 3(b) emulates the AGMNC implementation, which takes into account the latency asymmetry. This results in an implementation with only $2(d+1)$ additional registers and a latency of 3 cycles. Comparing these two implementations reveals that Fig. 3(b) requires 50% fewer registers and 25% less latency than Fig. 3(a). It can be seen that utilizing the latency asymmetry feature can lead to significant improvements in terms of both area and latency.



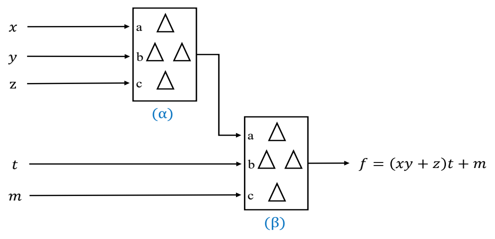
(a) The implementation by AGEMA.



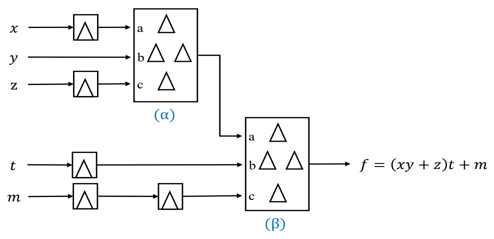
(b) The implementation by AGMNC.

Figure 3: Functionality xyz implementations using HPC-AND.

The latency asymmetry feature can be effectively utilized in the implementation of a Boolean function, as demonstrated by the following example using $f = (xy + z)t + m$. Firstly, we connect the later arriving signal to the input port of the gadget with the shorter latency, i.e., connecting the output of the AND-XOR gadget (marked as α) to the input port "a" of the other AND-XOR gadget (marked as β), instead of the input port "b" in Fig. 4(a). However, since the AND-XOR gadget has multiple input ports, there are inevitably some input ports that do not satisfy the latency requirements, i.e., ports "a" and "c" of the AND-XOR gadget (marked as α) and port "b" and "c" of the AND-XOR gadget (marked as β) in Fig. 4(a). In such scenarios, inserting registers in the input path is a viable solution to satisfy the latency constraints. Therefore, Fig. 4(b) presents the final implementation of the delay asymmetry example using the solution above, i.e., $5(d + 1)$ registers are inserted in the input path.



(a) The latency requirements are not met.



(b) The latency requirements are met.

Figure 4: The example of latency asymmetry of AND-XOR gadget.

To the best of our knowledge, we are the first work to integrate the latency asymmetry feature into an automated tool for generating masking schemes. Although a tool was also developed to exploit this feature, [CGLS20] focused on finding a circuit representation and did not automatically translate the circuit representation into a masked circuit. In addition to supporting the new AND-XOR gadget, AGMNC has developed an automated procedure to generate masked circuit integrating the latency asymmetry feature.

3.3 Implementation Optimization

Until now, we have presented two key techniques employed by AGMNC, namely the AND-XOR gadget and its latency asymmetry, which result in most cases in a final S-box design with considerably lower area or latency compared to the design generated by AGEMA. To further optimize our design, we present an implementation optimization technique in this section.

As elaborated in Sect. 3.2, if the latency requirements cannot be met by all input ports of the HPC-AND or AND-XOR gadget, insertion of registers in the input path becomes necessary to synchronize latency. Therefore, the number of registers inserted to synchronize latency directly impacts the area of the final S-box design. Further, we propose an efficient implementation technique for optimizing the number of registers required for synchronization latency. We illustrate two synchronization methods using the representation of SKINNY S-box[2] as an example: one is the common method without optimization (shown in Fig. 5), and the other one is an optimized method taken by our tool (shown in Fig. 6). In this example there are three layers, where $\mathcal{L}_0$ and $\mathcal{L}_1$ are nonlinear and $\mathcal{F}$ is linear. The t_0, t_1, t_2 and t_3 are the outputs of AND-XOR gadgets, while l_0, l_1, l_2, l_3 and l_4 are the outputs of linear operations.

As shown in Fig. 5, we depict the common implementation without optimization, where six, four, and zero different registers inserted in the $\mathcal{L}_0, \mathcal{L}_1$ and $\mathcal{F}$ layers, respectively, to synchronize latency. The process of synchronization latency can be summarized as follows: first, identify signals that fail to meet the required latency (i.e., $x_1+1, x_3, x_3+1, x_0, x_0+x_3$ and x_2 in layer $\mathcal{L}_0$, l_1, l_2, t_1 and t_0 in layer $\mathcal{L}_1$), and then insert some registers in the input path of these signals until the latency constraints are satisfied (i.e., insert a register in the input path of each of the above signals).

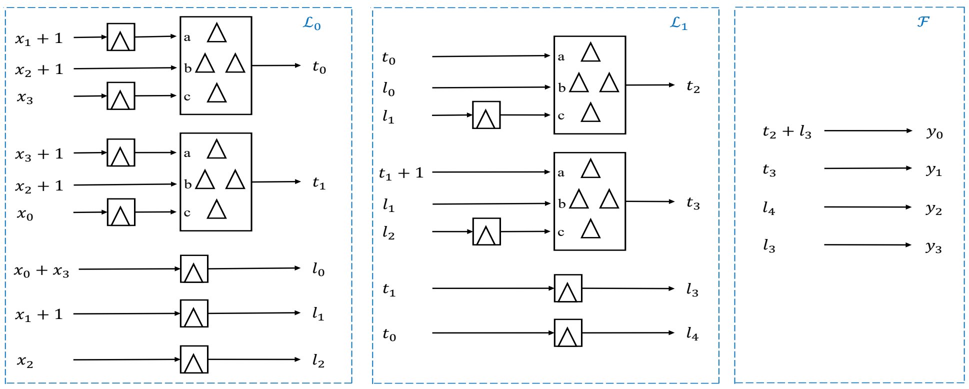


Figure 5: Synchronization without optimization.

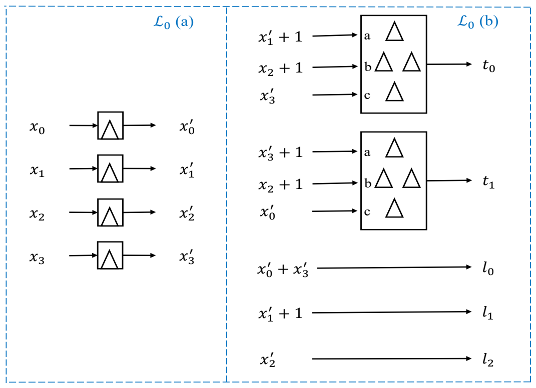


Figure 6: Synchronization with optimization.

[2]The detail of the representation for SKINNY S-box is shown in the supplementary material of this article.

After observing and analyzing the above implementation, we note that in layer $\mathcal{L}_0$, the signals that do not satisfy the latency requirements are generated using linear operations of the four primary inputs (i.e., x_0, x_1, x_2 and x_3). Additionally, some of these signals share a common primary input. More specifically, signals x_2+1 and x_2 share the primary input x_2, signals x_3, x_3+1 and x_0+x_3 share the primary input x_3, and signals x_0 and x_0+x_3 share the primary input x_0. Since the number of these signals is larger than the total number of primary inputs used to generate them (i.e., $6>4$), we propose a synchronization method that involves the insertion of registers in the path of the primary inputs. By doing so, we can ensure that the latency requirements of these primary inputs are met. Once this is achieved, we can use these already synchronized primary inputs to generate those signals that do not meet their latency requirements yet. This approach allows us to realize the synchronization process with fewer registers and to ensure that all signals in the circuit are correctly synchronized. Specifically, in the example of the SKINNY S-box, we insert registers in the path of each primary input (i.e., x_0, x_1, x_2 and x_3) until the latency constraints are satisfied, obtaining signals x'_0, x'_1, x'_2 and x'_3 at layer $\mathcal{L}_0(a)$ of Fig. 6. Then, at layer $\mathcal{L}_0(b)$, we assign x'_0, x'_1,x'_2 and x'_3 to the signals, the latency of which does not meet the requirements yet. At layer $\mathcal{L}_0$, this new approach requires only four registers to synchronize latency, saving two registers compared to the approach of AGEMA shown in Fig. 5. The $\mathcal{L}_1$ layer in the example exhibits a one-to-one correspondence between the unsynchronized signals and the primary inputs to that layer (i.e., l_1, l_2, t_1 and t_0). This implies that the number of unsynchronized signals in this layer is the same as the number of primary input signals. As a result, the remaining layers behave the same as Fig. 5. It should be noted that due to the masking scheme, i.e., each signal in Figs. 5 and 6 consists of at least $d+1$ shares, our implementation technique actually saves $2(d+1)$ registers in the above example, where d is the security order.

3.4 Automation Tool AGMNC

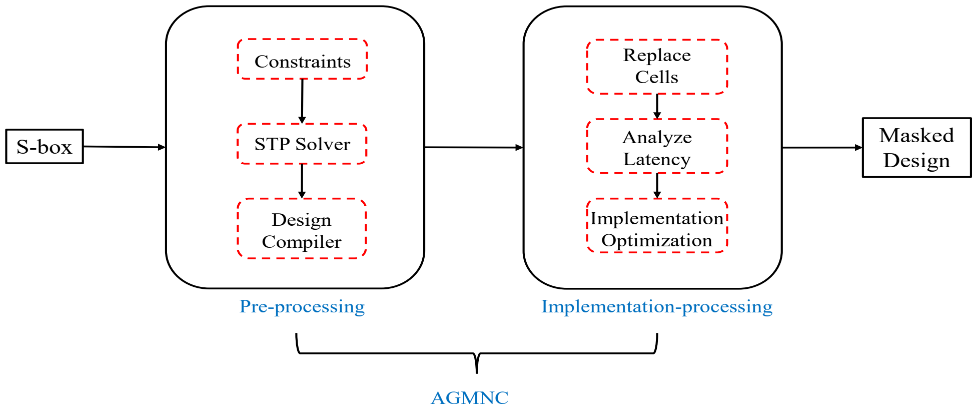


Figure 7: The operation process of automation tool AGMNC.

Fig. 7 shows the principle of the implementation of the automation tool AGMNC. The input of AGMNC is the look-up table description of nonlinear components (e.g., an S-box), and the output is a hardware masked implementation. The operation process of AGMNC consists of two phases: one is the pre-processing and the other one is the implementation-processing. In the pre-processing phase, AGMNC first finds a circuit representation suitable for our techniques and tool above for a given S-box using several constraints and the STP solver described in the next paragraph. Then, this circuit representation is synthesized into the corresponding netlist through the Design Compiler [Inc]. The unprotected netlist is fed to the implementation-processing module. For the implementation-processing phase, the first step is to extract and replace cells. This involves replacing the AND-XOR and AND gates in the netlist with the AND-XOR and HPC-AND[3] gadgets, respectively. The next step is calculating the latency for each input and output port of each AND-XOR and HPC-AND gadget. The latency asymmetry feature is checked and confirmed in this step. Subsequently, the implementation optimization technique is executed to synchronize the internal and output signals, the latency of which does not meet the requirements. The key techniques in the implementation-processing phase have been detailed in Sects. 3.1 to 3.3, hence this section focuses on the pre-processing phase.

Given the inherent complexity of searching for the circuit representation of a particular S-box, we utilize a solver based on the Boolean satisfiability (SAT) problem, namely STP, to find the circuit representation. As mentioned above, in the pre-processing phase, we should add several constraints to the STP solver to find the circuit representation suitable for our techniques and tool. The meaning and necessity of each constraint are described below.

Constraint 1 (the number of layers of AND-XOR and HPC-AND gadgets). Since the latency of each AND-XOR or HPC-AND gadget is two cycles, while the linear operations can be executed without latency, we control the latency of the final design by constraining the number of layers, i.e., the depth of AND-XOR and AND gadgets.

Constraint 2 (the number of AND-XOR and HPC-AND gadgets). Since the area of a single AND-XOR or HPC-AND gadget is significantly higher than that of a linear operation, especially for high-security orders, it is crucial to constrain the number of AND-XOR and AND gadgets to construct a final design with excellent area.

Constraint 3 (latency asymmetry feature). Considering the latency asymmetry of AND-XOR and AND gadgets, we have to add constraints so that the "b" input port of each AND-XOR and HPC-AND gadget (as shown in Figs. 3 and 4) is assigned to the signal generated by the linear operations of primary inputs or by the linear operations in the previous layer.

Constraint 4 (the number of unique signals at specific positions). This constraint is necessary to take advantage of the implementation optimization technique and further reduce the area of the final design. The unique signals are the primary inputs and the linear outputs in the previous layer. The specific positions are the "a" and "c" input ports of each AND-XOR gadget, the "a" input port of each HPC-AND gadget, each linear output, and the final output.

More specifically, Eq. (1) is a circuit representation of SKINNY S-box found by the STP solver. Eq. (1) consists of two layers of AND-XOR gadget and one final output layer. Two AND-XOR gadgets are in each AND-XOR gadget layer (i.e., t_0, t_1, t_2 and t_3), respectively. There are three, two linear outputs in each AND-XOR gadget layer (i.e., l_0, l_1, l_2, l_3 and l_4), respectively. The underlined terms are exactly the specific positions in each layer, with 8 unique variables (i.e., $x_0, x_1, x_2, x_3, l_1, l_2, l_3$ and l_4). In other words, at least $8(d+1)$ registers need to be inserted to synchronize the latency, where d is the security order.

[3]In some S-boxes, we find experimentally that using AND-XOR and HPC-AND gadgets together can result in a lower hardware cost than only one of these gadgets, such as the PRINCE S-box shown in the supplementary material of this article. Therefore, in addition to the AND-XOR gadget, AGMNC supports the HPC-AND gadget.

$$
\begin{aligned}
&S = \mathcal{F} \circ \mathcal{L}_1 \circ \mathcal{L}_0 \\
&\mathcal{L}_0 : t_0 = \underline{(x_1+1)}(x_2+1) + \underline{x_3}, \quad t_1 = \underline{(x_3+1)}(x_2+1) + \underline{x_0}, \quad l_0 = \underline{x_0 + x_3}, \\
&\qquad l_1 = \underline{x_1+1}, \quad l_2 = \underline{x_2} \\
&\mathcal{L}_1 : t_2 = \underline{t_0} l_0 + \underline{l_1}, \quad t_3 = \underline{(t_1+1)} l_1 + \underline{l_2}, \quad l_3 = \underline{t_1}, \quad l_4 = \underline{t_0} \\
&\mathcal{F} : y_0 = \underline{t_2 + l_3}, \quad y_1 = \underline{t_3}, \quad y_2 = \underline{l_4}, \quad y_3 = \underline{l_3}.
\end{aligned} \tag{1}
$$

4 Case Studies

This section provides several S-box and full cipher implementations to highlight the benefits of applying our techniques and tool from Sect. 3.

4.1 S-boxes

Following [Sto16, LWH+21], we encode the constraints above described in Sect. 3.4 into the STP solver. As a result, we find several circuit representations for the 4-bit S-boxes of SKINNY [BJK+16], PRESENT [BKL+07], PRINCE [BCG+12] and Midori [BBI+15]. Further, we also adapt the design in [BP12] to reconstruct the 8-bit S-box of AES [DR99], in which two parts are modified, one is the inversion in $GF(2^4)$ (seen as a 4-bit S-box by the STP solver), and the other one is the generation of the inputs signals for the inversion in $GF(2^4)$. In particular, we reduce an AND gate in the inversion in $GF(2^4)$ compared to the design from [BP12], further reducing some randomness and hardware area required by the masked AND gadget. Note that all S-box representations are presented in the supplementary material of this article, where each representation consists of three layers, i.e., $\mathcal{L}_0, \mathcal{L}_1$ are the nonlinear layers, and $\mathcal{F}$ is the linear layer.

To compare our work to state of the art, we provide two different masking schemes, one is generated by AGMNC, employing the designs found by STP solver, and the other one is generated by AGEMA, referring to [CGLS20, BP12]. In this section, we use Synopsys Design Compiler R-2020.09-SP4 and NanGate 45 nm standard cell library to synthesize the masking schemes of different S-boxes. Tables 1 and 2 list the synthesized experimental results of the SKINNY S-box and AES S-box; the performance of other S-boxes can be found in the supplementary material of this article.

From the synthesized experimental results, our work significantly reduces the area overhead compared to AGEMA. Specifically, we achieve an area reduction of about $21\% \sim 41\%$ using AND-XOR1 gadgets and about $13\% \sim 34\%$ using AND-XOR2 gadgets. Notably, for the first-order secure SKINNY S-box, the reduction in area is approximately 41% and 34% using AND-XOR1 and AND-XOR2 gadgets, respectively. The area reduction can be attributed to the utilization of AND-XOR gadgets and to two key techniques, i.e., the latency asymmetry and implementation optimization. Regarding latency, our work outperforms AGEMA by 25%, and specifically, for the AES Sbox, we reduce the latency from eight cycles to six cycles. The latency asymmetry feature of AND-XOR and HPC-AND gadgets leads to a latency reduction. In addition, compared to AGEMA, our work reduces the dynamic power of about $16\% \sim 49\%$ and about $23\% \sim 47\%$ using AND-XOR1 and AND-XOR2 gadgets, respectively. In particular, the dynamic power reduction for the first-order secure SKINNY S-box is approximately 49% and 47% using AND-XOR1 and AND-XOR2 gadgets, respectively. These results demonstrate that our proposed techniques and tool not only greatly reduce the area and latency overhead, but also significantly reduce the dynamic power.

Table 1: Hardware performance figures of the SKINNY S-box.

Masking Scheme	Order	Area [GE]	Latency [cycle]	Power [uW]	Rand. [bit]	Delay [ns]	Ref.
HPC1	1	658	4	5.39	8	0.34	[KMMS21]
	2	1156	4	10.00	20	0.39	[KMMS21]
	3	1733	4	15.90	40	0.39	[KMMS21]
	4	2379	4	22.52	60	0.45	[KMMS21]
HPC2	1	785	4	6.79	4	0.39	[KMMS21]
	2	1552	4	14.17	12	0.45	[KMMS21]
	3	2570	4	24.08	24	0.50	[KMMS21]
	4	3839	4	36.56	40	0.54	[KMMS21]
AND-XOR1	1	**385** (↓41%)	**3** (↓25%)	**2.75** (↓49%)	8	0.30	This Work
	2	**747**	**3**	**5.81**	20	0.36	This Work
	3	**1187**	**3**	**10.10**	40	0.37	This Work
	4	**1696**	**3**	**15.20**	60	0.44	This Work
AND-XOR2	1	**517** (↓34%)	**3** (↓25%)	**3.61** (↓47%)	4	0.36	This Work
	2	**1151**	**3**	**8.65**	12	0.43	This Work
	3	**2035**	**3**	**15.74**	24	0.48	This Work
	4	**3169**	**3**	**15.74**	40	0.54	This Work

Table 2: Hardware performance figures of the AES S-box.

Masking Scheme	Order	Area [GE]	Latency [cycle]	Power [uW]	Rand. [bit]	Delay [ns]	Ref.
HPC1	1	4263	8	47.28	68	0.49	[KMMS21]
	2	7840	8	85.81	170	0.54	[KMMS21]
	3	12085	8	133.05	340	0.54	[KMMS21]
	4	16920	8	188.31	510	0.60	[KMMS21]
HPC2	1	5340	8	61.43	34	0.60	[KMMS21]
	2	11206	8	133.96	102	0.76	[KMMS21]
	3	19203	8	231.31	204	0.91	[KMMS21]
	4	29330	8	358.36	340	1.04	[KMMS21]
AND-XOR1	1	**2895** (↓32%)	**6** (↓25%)	**31.28** (↓34%)	**66** (↓3%)	0.49	This Work
	2	**5745**	**6**	**63.54**	**165**	0.57	This Work
	3	**9243**	**6**	**106.06**	**330**	0.64	This Work
	4	**13314**	**6**	**157.48**	**495**	0.74	This Work
AND-XOR2	1	**3967** (↓26%)	**6** (↓25%)	**42.90** (↓30%)	**33** (↓3%)	0.55	This Work
	2	**9078**	**6**	**97.33**	**99**	0.69	This Work
	3	**16239**	**6**	**173.77**	**198**	0.78	This Work
	4	**25469**	**6**	**274.26**	**330**	0.91	This Work

4.2 Full Ciphers

For the full ciphers, we provide three case studies: SKINNY, which is round-based encryption, PRESENT, which is nibble-serial encryption, and AES, which is byte-serial encryption. The above cases refer to designs from [BJK+16, BKL+07, DR02], respectively.

To ensure a fair comparison, we initially employ AGEMA to generate the masking scheme of the above ciphers and then replace its S-boxes with our constructions. Tables 3 and 4 list the hardware performance of SKINNY and AES; the performance of PRESENT can be found in the supplementary material of this article. Although our work focuses on S-box implementations, we have also made comparable performance in the full ciphers. Specifically, we achieve approximately 18% ∼ 27% (resp., 17% ∼ 24%), 20% ∼ 22% (resp., 20% ∼ 22%) and 18% ∼ 32% (resp., 19% ∼ 31%) reduction in area, latency and dynamic power using AND-XOR1 gadgets (resp., AND-XOR2 gadgets), respectively. It is worth mentioning that compared to the current automated tool AGEMA, for the first order masking schemes of the SKINNY round-based encryption, we achieve a reduction of approximately 27% (resp., 24%) in the area and 20% (resp., 20%) in latency using AND-XOR1 gadgets (resp., AND-XOR2 gadgets), respectively. Due to the reduction of an AND gate in the representation of the AES S-box, we reduce the randomness by 2 (resp., 1), 5 (resp., 3), and 10 (resp., 6) bits in the AES byte-serial encryption with security order 1, 2 and 3 using AND-XOR1 gadgets (resp., AND-XOR2 gadgets), respectively. Whether the security order is 1, 2, or 3, and whether using AND-XOR1 or AND-XOR2 gadgets, we achieve the reduction of approximately 20% in the area and 22% in latency, respectively. The reduction realized by our automation tool AGMNC almost bridge the gap between a hand-crafted implementation [MCS22] and the current automation tool AGEMA [KMMS21].

Table 3: Hardware performance figures of the SKINNY round-based encryption function.

Masking Scheme	Order	Area [GE]	Latency [cycle]	Power [uW]	Rand. [bit]	Delay [ns]	Ref.
HPC1	1	18855	165	191.98	128	0.57	[KMMS21]
	2	30759	165	323.93	320	0.62	[KMMS21]
	3	43931	165	459.97	640	0.62	[KMMS21]
HPC2	1	20881	165	221.47	64	0.62	[KMMS21]
	2	37095	165	409.09	192	0.67	[KMMS21]
	3	57328	165	646.12	384	0.72	[KMMS21]
AND-XOR1	1	**13694** (↓27%)	**132** (↓20%)	**129.83** (↓32%)	128	0.45	This Work
	2	**23049**	**132**	**228.17**	320	0.50	This Work
	3	**33663**	**132**	**334.16**	640	0.50	This Work
AND-XOR2	1	**15806** (↓24%)	**132** (↓20%)	**151.74** (↓31%)	64	0.50	This Work
	2	**29513**	**132**	**292.81**	192	0.56	This Work
	3	**47231**	**132**	**476.81**	384	0.60	This Work

Table 4: Hardware performance figures of the AES byte-serial encryption function.

Masking Scheme	Order	Area [GE]	Latency [cycle]	Power [uW]	Rand. [bit]	Delay [ns]	Ref.
HPC1	1	39318	2043	396.84	68	1.24	[KMMS21]
	2	59794	2043	600.98	170	1.42	[KMMS21]
	3	80946	2043	820.55	340	1.94	[KMMS21]
HPC2	1	40395	2043	409.65	34	1.29	[KMMS21]
	2	63161	2043	662.63	102	1.42	[KMMS21]
	3	88050	2043	930.67	204	1.48	[KMMS21]
AND-XOR1	1	**30969** (↓21%)	**1589** (↓22%)	**306.53** (↓23%)	**66** (↓3%)	1.22	This Work
	2	**47362**	**1589**	**477.11**	**165**	1.49	This Work
	3	**64393**	**1589**	**651.54**	**330**	1.60	This Work
AND-XOR2	1	**32041** (↓21%)	**1589** (↓22%)	**324.74** (↓21%)	**33** (↓3%)	1.33	This Work
	2	**50695**	**1589**	**521.34**	**99**	1.49	This Work
	3	**71409**	**1589**	**732.92**	**198**	1.88	This Work

5 Security Analysis

5.1 Theoretical Evaluation

Composable security is a crucial concept in the masking scheme, as it ensures that the level of security of a composed circuit is maintained if its standalone secure sub-circuits meet certain requirements. To our knowledge, PINI is widely used as the most efficient method for defining these requirements. More specifically, if several sub-circuits, also known as gadgets, meet the concept of PINI under the glitch-extended probing model, and the connections of these sub-circuits do not intermix share domains, the overall composed circuit is also PINI secure under the glitch-extended probing model. It is essential to guarantee that the output shares of a gadget are connected with the input shares of another gadget in the same share indexes. For example, if (y_0, y_1) and (x_0, x_1) are the two output shares of a gadget and the two input shares of another gadget, respectively, the only available connection is $(x_0, x_1) = (y_0, y_1)$.

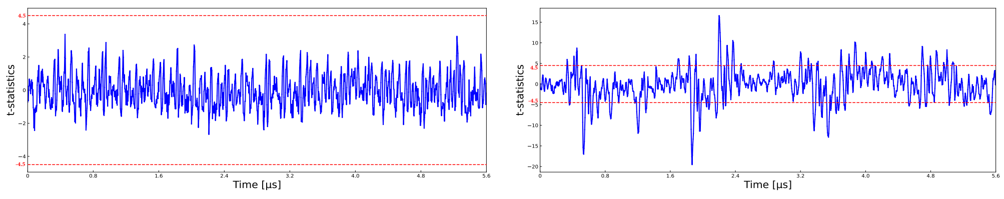
To ensure the sub-circuits are PINI secure under the glitch-extended probing model, we have examined the implementations of all sub-circuits with the SILVER tool [KSM20], including HPC1, HPC2, AND-XOR1, AND-XOR2, NOT_masked, XOR_masked and XNOR_masked. Further, the validity of the connections is guaranteed because AGMNC uses deterministic procedures to realize the connections between sub-circuits. In addition to the above analysis, for the masked S-boxes described in this article, we confirm the security of all 4-bit S-boxes and some AES S-boxes under the glitch-extended probing model using SILVER.

Given that the SILVER is currently incapable of analyzing full cipher implementations, we opted to perform experimental analysis. To this end, we implemented full ciphers with our masked S-boxes on the FPGA evaluation board and collected power consumption traces. The results are presented in the next section.

5.2 Experimental Evaluation

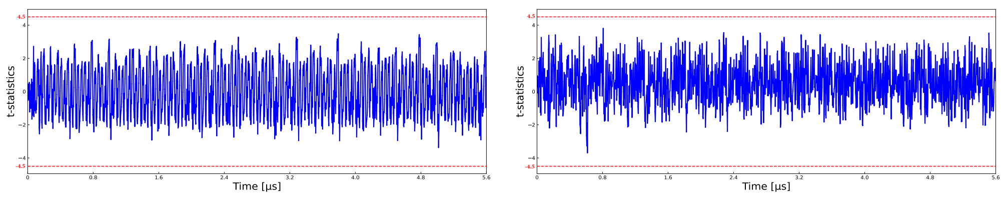
We implement the full ciphers on a SAKURA-G [SAK] board, where a Spartan-6 FPGA is embedded for practical SCA evaluations. Our designs utilized in this section are provided with a stable clock signal at the frequency of 24 MHz. The power consumption traces of the above board are monitored with a PicoScope 5244D oscilloscope at a sampling frequency of 250 MS/s. Each randomness bit is dynamically generated during runtime with the use of a 31-bit maximum length Linear Feedback Shift Register (LFSR), with feedback polynomial $x^{31} + x^{28} + 1$[4]. Then, each LFSR is initialized with an independent random seed. As the leakage assessment scheme, we perform fixed-versus-random t-test [GGJR+11], which is widely used to evaluate the security of masked implementations. In each design, we keep the key constant and collect 1 million traces to conduct the t-test analysis.

Figs. 8 and 9 depict the t-test results of SKINNY based on AND-XOR2 for the first-order and second-order masking schemes, respectively. The results in Figs. 8(a), 9(a) and 9(b) confirm the first-order security for the first-order design, as well as the first- and second-order security for the second-order design. For comparison, we also provide the results in Figs. 8(b) and 9(c) for the two designs when PRNG OFF to check our measurement setup.

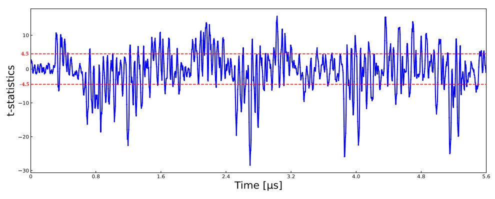


(a) PRNG ON, 1st-order t-test. (b) PRNG OFF, 1st-order t-test.

Figure 8: SKINNY round-based encryption, first-order based on AND-XOR2.



(a) PRNG ON, 1st-order t-test. (b) PRNG ON, 2nd-order t-test.



(c) PRNG OFF, 1st-order t-test.

Figure 9: SKINNY round-based encryption, second-order based on AND-XOR2.

[4]This LFSR design is one of the FPGA-optimized designs described in [Alf98].

6 Conclusions

In this article, we developed a user-friendly tool for the automated generation of masked nonlinear components (AGMNC), which enables hardware designers, regardless of their experience level, to elegantly and efficiently create secure masked hardware S-box circuits starting from the look-up table description of an S-box.

AGMNC utilizes the AND-XOR gadget, latency asymmetry feature, and implementation optimization to generate efficient masked circuits. Furthermore, we show how to find implementations of given S-boxes that satisfy our techniques using the STP solver. We use AGMNC to generate a masked implementation of several S-boxes and evaluate the hardware performance of the masked implementation from the perspective of the individual S-box and the full cipher. The evaluation shows that our designs require less area, latency, dynamic power, and even randomness, with a reduction of up to 41%, 25%, 49%, and 3% than AGEMA, respectively. Finally, we use the SILVER tool and FPGA-based practical experiments to verify the security of our designs. Based on the above results, we believe that AGMNC is an elegant and efficient automated tool to generate secure masked designs, with potential applications in various hardware design and implementation scenarios.

As an open problem, it is important to acknowledge that the hardware performance of the final designs generated based on the composability notion of PINI is less efficient in terms of area, latency, and randomness compared to the hand-crafted designs described in related studies, e.g., [SM21a, SM21b, Sug19, GMK16]. This finding motivates the need for additional composable gadgets and implementation optimization techniques to improve the hardware performance. We believe that future research can focus on exploring new techniques and methods to address these challenges and achieve optimal hardware performance for secure masked cryptographic hardware circuits.

References

[Alf98] Peter Alfke. Efficient shift registers, LFSR counters, and long pseudo-random sequence generators. 1998. https://docs.xilinx.com/v/u/en-US/xapp052.

[BBD$^+$15] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. Verified proofs of higher-order masking. In *Advances in Cryptology–EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, pages 457–485. Springer, 2015.

[BBD$^+$16] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong non-interference and type-directed higher-order masking. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 116–129, 2016.

[BBI$^+$15] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita, and Francesco Regazzoni. Midori: A block cipher for low energy. In *Advances in Cryptology–ASIACRYPT 2015: 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29–December 3, 2015, Proceedings, Part II 21*, pages 411–436. Springer, 2015.

[BCG$^+$12] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, et al. PRINCE–a low-latency block cipher for pervasive computing applications. In *Advances in Cryptology–ASIACRYPT 2012: 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings 18*, pages 208–225. Springer, 2012.

[BDF$^+$17] Gilles Barthe, François Dupressoir, Sebastian Faust, Benjamin Grégoire, FrançoiS-Xavier Standaert, and Pierre-Yves Strub. Parallel implementations of masking schemes and the bounded moment leakage model. In *Advances in Cryptology–EUROCRYPT 2017: 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30–May 4, 2017, Proceedings, Part I 36*, pages 535–566. Springer, 2017.

[BJK$^+$16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY family of block ciphers and its low-latency variant MANTIS. In *Advances in Cryptology–CRYPTO 2016: 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II 36*, pages 123–153. Springer, 2016.

[BKL$^+$07] Andrey Bogdanov, Lars R Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew JB Robshaw, Yannick Seurin, and Charlotte Vikkelsoe. PRESENT: An ultra-lightweight block cipher. In *Cryptographic Hardware and Embedded SystemS-CHES 2007: 9th International Workshop, Vienna, Austria, September 10-13, 2007. Proceedings 9*, pages 450–466. Springer, 2007.

[BP12] Joan Boyar and René Peralta. A small depth-16 circuit for the AES S-Box. In *Information Security and Privacy Research: 27th IFIP TC 11 Information*

Security and Privacy Conference, SEC 2012, Heraklion, Crete, Greece, June 4-6, 2012. Proceedings 27, pages 287–298. Springer, 2012.

[CGLS20] Gaëtan Cassiers, Benjamin Grégoire, Itamar Levi, and FrançoiS-Xavier Standaert. Hardware private circuits: From trivial composition to full verification. *IEEE Transactions on Computers*, 70(10):1677–1690, 2020.

[CJRR99] Suresh Chari, Charanjit S Jutla, Josyula R Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In *Advances in Cryptology—CRYPTO'99: 19th Annual International Cryptology Conference Santa Barbara, California, USA, August 15–19, 1999 Proceedings 19*, pages 398–412. Springer, 1999.

[CS20] Gaëtan Cassiers and FrançoiS-Xavier Standaert. Trivially and efficiently composing masked gadgets with probe isolating non-interference. *IEEE Transactions on Information Forensics and Security*, 15:2542–2555, 2020.

[DDF14] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models: from probing attacks to noisy leakage. In *Advances in Cryptology–EUROCRYPT 2014: 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings 33*, pages 423–440. Springer, 2014.

[DR99] Joan Daemen and Vincent Rijmen. AES proposal: Rijndael. 1999.

[DR02] Joan Daemen and Vincent Rijmen. *The design of Rijndael, 2nd edition.* Springer, 2002.

[FGP$^+$18] Sebastian Faust, Vincent Grosso, SMD Pozo, Clara Paglialonga, and F-X Standaert. Composable masking schemes in the presence of physical defaults & the robust probing model. 2018.

[GGJR$^+$11] Benjamin Jun Gilbert Goodwill, Josh Jaffe, Pankaj Rohatgi, et al. A testing methodology for side-channel resistance validation. In *NIST non-invasive attack testing workshop*, volume 7, pages 115–136, 2011.

[GM18] Hannes Groß and Stefan Mangard. A unified masking approach. *Journal of cryptographic engineering*, 8:109–124, 2018.

[GMK16] Hannes Groß, Stefan Mangard, and Thomas Korak. Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. *Cryptology ePrint Archive*, 2016.

[GMK17] Hannes Groß, Stefan Mangard, and Thomas Korak. An efficient side-channel protected AES implementation with arbitrary protection order. In *Topics in Cryptology–CT-RSA 2017: The Cryptographers' Track at the RSA Conference 2017, San Francisco, CA, USA, February 14–17, 2017, Proceedings*, pages 95–112. Springer, 2017.

[GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In *Cryptographic Hardware and Embedded Systems—CHES 2001: Third International Workshop Paris, France, May 14–16, 2001 Proceedings 3*, pages 251–261. Springer, 2001.

[HS14] Michael Hutter and Jörn-Marc Schmidt. The temperature side channel and heating fault attacks. In *Smart Card Research and Advanced Applications: 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers 12*, pages 219–235. Springer, 2014.

[Inc] Synopsys Inc. Design compiler graphical. https://www.synopsys.com.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In *Advances in Cryptology-CRYPTO 2003: 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003. Proceedings 23*, pages 463–481. Springer, 2003.

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Advances in Cryptology—CRYPTO'99: 19th Annual International Cryptology Conference Santa Barbara, California, USA, August 15–19, 1999 Proceedings 19*, pages 388–397. Springer, 1999.

[KMMS21] David Knichel, Amir Moradi, Nicolai Müller, and Pascal Sasdrich. Automated generation of masked hardware. *Cryptology ePrint Archive*, 2021.

[Koc96] Paul C Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Advances in Cryptology—CRYPTO'96: 16th Annual International Cryptology Conference Santa Barbara, California, USA August 18–22, 1996 Proceedings 16*, pages 104–113. Springer, 1996.

[KSM20] David Knichel, Pascal Sasdrich, and Amir Moradi. SILVER–statistical independence and leakage verification. In *Advances in Cryptology–ASIACRYPT 2020: 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7–11, 2020, Proceedings, Part I 26*, pages 787–816. Springer, 2020.

[LWH$^+$21] Zhenyu Lu, Weijia Wang, Kai Hu, Yanhong Fan, Lixuan Wu, and Meiqin Wang. Pushing the limits: Searching for implementations with the smallest area for lightweight S-boxes. In *Progress in Cryptology–INDOCRYPT 2021: 22nd International Conference on Cryptology in India, Jaipur, India, December 12–15, 2021, Proceedings 22*, pages 159–178. Springer, 2021.

[MCS22] Charles Momin, Gaëtan Cassiers, and FrançoiS-Xavier Standaert. Handcrafting: Improving Automated Masking in Hardware with Manual Optimizations. In *Constructive Side-Channel Analysis and Secure Design: 13th International Workshop, COSADE 2022, Leuven, Belgium, April 11-12, 2022, Proceedings*, pages 257–275. Springer, 2022.

[SAK] SAKURA. Side-channel Attack User Reference Architecture. http://satoh.cs.uec.ac.jp/SAKURA/index.html.

[SM21a] Aein Rezaei Shahmirzadi and Amir Moradi. Re-consolidating first-order masking schemes: Nullifying fresh randomness. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 305–342, 2021.

[SM21b] Aein Rezaei Shahmirzadi and Amir Moradi. Second-order SCA security with almost no fresh randomness. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 708–755, 2021.

[Sto16] Ko Stoffelen. Optimizing S-box implementations for several criteria using SAT solvers. In *Fast Software Encryption: 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers 23*, pages 140–160. Springer, 2016.

[Sug19] Takeshi Sugawara. 3-share threshold implementation of AES S-box without fresh randomness. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 123–145, 2019.