IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2024, No. 3, pp. 249–272. DOI:10.46586/tches.v2024.i3.249-272

Time Sharing - A Novel Approach to Low-Latency Masking

Dilip Kumar S. V.[1], Siemen Dhooghe[1], Josep Balasch[2], Benedikt Gierlichs[1] and Ingrid Verbauwhede[1]

[1] COSIC, ESAT, KU Leuven, Leuven, Belgium
[2] e-Media Research Lab, STADIUS, KU Leuven, Leuven, Belgium
{dshanmug,siemen.dhooghe,josep.balasch,benedikt.gierlichs,ingrid.verbauwhede}@esat.kuleuven.be

Abstract. We present a novel approach to small area and low-latency first-order masking in hardware. The core idea is to separate the processing of shares in time in order to achieve non-completeness. Resulting circuits are proven first-order glitch-extended PINI secure. This means the method can be straightforwardly applied to mask arbitrary functions without constraints which the designer must take care of. Furthermore we show that an implementation can benefit from optimization through EDA tools without sacrificing security. We provide concrete results of several case studies. Our low-latency implementation of a complete PRINCE core shows a 32% area improvement (44% with optimization) over the state-of-the-art. Our PRINCE S-Box passes formal verification with a tool and the complete core on FPGA shows no first-order leakage in TVLA with 100 million traces. Our low-latency implementation of the AES S-Box costs roughly one third (one quarter with optimization) of the area of state-of-the-art implementations. It shows no first-order leakage in TVLA with 250 million traces.

Keywords: Hardware · Masking · Probing Security · Side-Channel Analysis

1 Introduction

Implementing secure cryptographic algorithms in a computer system without compromising their promised security has always been challenging. Early research demonstrating the vulnerabilities of cryptographic implementations by Kocher *et al.* [Koc96] showed that it is possible to find the secrets that a computer processes by monitoring its execution time and thereby highlighted the need to build secure implementations. This led to the consolidation of side-channel analysis as a field of study that attempts to gain information from the implementation of a chip or computer system by monitoring its physical effects rather than exploiting a weakness of the implemented algorithm. Along with timing analysis [Koc96], power analysis [KJJ99] and electromagnetic analysis [GMO01, QS01] represent some of the best-known side-channel attacks. Power analysis, in particular, is perhaps the most popular due to its low setup cost, non-invasive nature, and devastating effectiveness.

In the past few decades, there has been a great deal of research on securing cryptographic implementations against side-channel attacks. Chari *et al.* [CJRR99] as well as Goubin and Patarin [GP99] independently proposed a generic countermeasure called masking that splits the data being processed into random shares to thwart power analysis attacks. The idea behind the countermeasure is to eliminate the correlation between the secret data and the data being processed, since the device's power consumption depends on the latter, which is now random. But processing multiple shares also comes with

Received: 2024-01-15 Accepted: 2024-03-15 Published: 2024-07-18

overheads in terms of implementation area, execution time, online randomness, etc. Along with securing implementations, it has also been important to minimize these overheads. Masking proved to be quite successful for securing software implementations, but it was later found that masked hardware implementations still leak information about the secrets due to glitches [MPG05]. Several modern masking techniques such as Threshold Implementations (TI) [NRR06], Consolidating Masking Schemes (CMS) [RBN+15], and Domain Oriented Masking (DOM) [GMK16] were proposed to securely mask hardware in the presence of glitches. They were quite successful in creating secure and efficient hardware implementations with low area and randomness usage. Overhead reductions are typically achieved by decomposing complex non-linear functions, such as an S-Box, into smaller sub-circuits with low algebraic degrees that can be masked efficiently. The composition of the sub-circuits requires careful use of register stages to prevent glitch propagation and re-masking intermediate values to maintain uniformity.

Due to the recent advent of IoT devices, which are very accessible to an attacker, there is a need for embedded real-time applications to have fast data processing, such as memory encryption, to ensure security. As a consequence, there is a new motivation to design masking schemes suitable for low-latency implementations. One of the first generic approaches called GLM was proposed by Groß *et al.* and was used to design low-latency S-Boxes in [GIB18]. GLM, built upon DOM, reduces latency by eliminating register stages required for share compression after non-linear operations. Skipping share compression exponentially increases the share count after every non-linear operation, drastically increasing the overall area and randomness utilization. This especially makes the approach impractical for masking large functions such as higher-degree S-Boxes. Other research into low-latency masking includes LLTI [AZN21] based on TI and other methods involving asynchronous circuits [MS16, NGPM22].

Although masking techniques are typically proven secure in the t-probing model [ISW03], most are not generic and are not trivial to compose with other design elements. In other words, converting any unprotected circuit to a protected one is not straightforward and is usually laborious. Recently in [CS20], Cassiers *et al.* introduced a new security notion called Probe Isolating Non-Interference (PINI), which allows for trivial composition. Any PINI gadget is directly composable with other (linear and non-linear) PINI gadgets, without significant overheads. In [CGLS21], the authors propose two small multiplication gadgets called HPC1 and HPC2 that can be composed in the glitch-extended probing model, introduced by Faust *et al.* [FGP+18], to create more complex circuits. Later in [KM22], Knichel *et al.* proposed the HPC3 gadget specifically intended for low-latency applications. Although one can build any circuit with these gadgets, the latency of the circuit grows with the algebraic degree of the function. To the best of our knowledge, there exists only one algorithm-level approach (which does not simply compose elementary gadgets) to generate first-order PINI secure circuits, namely GHPC [KSM22]. Despite being PINI secure, their low latency version GHPC_{LL} also suffers from the high area and randomness overheads for larger functions, like GLM. A single-cycle AES S-Box using GHPC_{LL} costs 64.1 kGEs and 2048 bits of randomness, similar to the cost of GLM. But GHPC_{LL} has the advantage that it is proven to be composable secure while GLM is not.

Contributions. We present a new masking method for low-latency applications that is first-order PINI composable secure, and - more importantly - brings substantially less overhead than other composable low-latency masking schemes. Our contributions are the following:

- We present a masking method that secures any function against first-order attacks and uses only a single register stage, thus executes in a single clock cycle.
- We provide a formal description and follow up with a proof that shows any circuit secured by our approach is first-order glitch-extended PINI secure.

- Compared to previously published algorithm-level approaches such as GLM [GIB18] and GHPC [KSM22] that implement single-cycle Boolean functions, our method shows a substantial improvement both in terms of area as well as online randomness required, for realistically complex circuits.

- We apply our proposed method to produce a masked first-order secure PRINCE implementation that executes in one cycle per round and show the improvements in the utilization results. We demonstrate the security of our PRINCE S-Box with a formal verification tool and show that the complete PRINCE core on FPGA exhibits no first-order leakage in TVLA with 100 million traces.

- We apply our proposed method to mask a more complex function, *i.e.* the AES S-Box, in order to demonstrate its potential for efficient implementations. We show significant improvements in utilization costs and demonstrate that our method scales well especially when masking larger functions. Our AES S-Box on FPGA shows no first-order leakage in TVLA with 250 million traces.

Outline. The remainder of this article is organized as follows. In Section 2, we briefly introduce the notation and recall relevant background. In Section 3, we start explaining our idea with a toy example and design a simple masked AND gate. Next, we detail our method with a formal description which can then be applied to any Boolean function. We prove that our method is first-order PINI glitch-extended composable secure. In Section 4 we discuss advantages of our approach. In Section 5, we present our first case study by applying our method to produce a masked, first-order secure, low-latency PRINCE core. In Section 6, we present our second case study by applying our method to a more complex and challenging function, the AES S-Box. In Section 7, we verify through leakage assessment tests that our theoretical proof does indeed translate to practically secure implementations of a complete PRINCE core and an AES S-Box. Finally, Section 8 presents our conclusions and avenues for future work.

2 Preliminaries

In this section we briefly introduce the notation and recall relevant background.

2.1 Notation

Boolean masking splits each bit $x \in \mathbb{F}_2$ into n uniform random shares x_i such that $x = x_0 \oplus \ldots \oplus x_{n-1}$. The storage of a variable in a register is denoted by curly brackets $\{\cdot\}$.

2.2 Probing Model

In the probing model, introduced by Ishai, Sahai, and Wagner [ISW03], an adversary $\mathcal{A}$ is allowed to observe a set of at most t (predefined) wires of a circuit at each execution of the masking. The security of a given implementation is proven by showing that a simulator $\mathcal{S}$ can perfectly simulate any set of at most t probes without any knowledge of the input shares $(x_0, \ldots, x_{n-1})$. A circuit ensuring this condition for any set of size t is said to be t-probing secure.

The probing model provides a way to prove the security of a circuit. However, this algorithmic circuit does not trivially map to practice where, due to leakage effects, an adversary can gain more information than what a probe typically captures. The main leakage effect to be considered is that of glitches. In this work, we model glitches by bundling groups of wires over which a glitch could carry information from one wire to another. Whereas one of the adversary's probes normally results in the value of a single wire, a glitch-extended probe allows obtaining the values of all wires in a bundle. This extension of the probing model has been discussed in the work of Reparaz et al. [RBN+15] and formalized by Faust et al. [FGP+18]. The formulation of the latter work is as follows: "For any ϵ-input circuit gadget G, combinatorial recombinations (aka glitches) can be modeled with specifically ϵ-extended probes so that probing any output of the function allows the adversary to observe all its ϵ inputs."

2.3 Related Work

Due to the growing interest in low-latency masking, several masking techniques have been developed in recent years which are specifically focused on reducing latency. Some techniques relevant to our work are GLM [GIB18], GHPC [KSM22] and LMDPL [SBHM20]. Among these, the constructions for GHPC and GLM are most comparable to our technique, while LMDPL employs a distinct dual-rail precharge logic.

GLM is a low-latency masking approach proposed by Groß *et al.* that can be applied to protect any security-sensitive circuit [GIB18]. GLM is based on the Domain Oriented-Masking (DOM) scheme [GMK16], which was introduced to create low-area and low randomness designs. DOM is a gate-level masking technique that uses masked AND gates (DOM multipliers) to build and secure more complex circuits. A masked circuit built with DOM is split into independent circuits called "domains" based on the share index of variables. Non-linear operations compute on all shares of a variable requiring communication between domains and are called "cross-domain terms". For a secure computation, the cross-domain terms are refreshed and stored in registers before they are compressed and merged with inner-domain terms to limit the number of shares and to reduce area.

While DOM optimizes for area and randomness, GLM trades area and randomness for reduced latency. The register stages in DOM multipliers increase latency in a design. GLM reduces latency by eliminating these stages. However, constructing a low-latency circuit by eliminating these register stages introduces complications, leading to increased area and randomness requirements for the circuit. First, the number of output shares increases after every non-linear operation as there is no share compression due to the lack of register stages. The cross-domain terms cannot be merged with the inner-domain terms, increasing the number of shares. Furthermore, as the non-linear logic depth increases, the number of shares of the intermediate values in the circuit also increases exponentially, resulting in a significant increase in area. Second, removing the registers causes the circuit to be susceptible to variable collisions. GLM requires the inputs to non-linear gates to be independently shared. If the circuit violates this condition, the colliding variables must be duplicated with multiple shared instances of the same variable with independent sharings. To resolve collisions, it might also be necessary to duplicate the entire fan-in circuitry causing the collision. All of these fixes increase the area and randomness overhead of the circuit. As a final step, secure share compression is performed by refreshing the shares with randomness and storing them in registers before the compression, which also increases the cost of area and randomness since many shares need to be refreshed and stored.

GHPC, introduced by Knichel *et al.* [KSM22], is a low-latency masking technique that uses Shannon Decomposition to transform arbitrary Boolean functions into secure PINI composable gadgets. For simplicity, we will illustrate the technique by applying it to a 4 × 4 function, $F(x, y, z, w) : \mathbb{F}_2^4 \rightarrow \mathbb{F}_2^4$. The technique is applied independently to each coordinate function, decomposing it into cofactors by fixing the input shares within a single share domain. Consider $f(x_0 + x_1, y_0 + y_1, z_0 + z_1, w_0 + w_1) : \mathbb{F}_2^4 \rightarrow \mathbb{F}_2$ as the shared representation of one of the four coordinate functions of F, where the subscript denotes the share domain of the inputs. The function f is decomposed into 16 cofactors by considering all combinations of inputs from the second share domain, *i.e.* $\{x_1, y_1, z_1, w_1\} \in \{0, 1\}^4$.

For example, if $\{x_1, y_1, z_1, w_1\} = \{0, 1, 1, 0\}$, then the corresponding cofactor would be $f(x_0, \overline{y_0}, \overline{z_0}, w_0)$. The resulting Shannon cofactors only depend on the inputs from the first share domain, *i.e.* $\{x_0, y_0, z_0, w_0\}$. A secure implementation of F with GHPC necessitates two register layers. A secure low-latency implementation of F with GHPC_{LL} reduces this to one register layer at the cost of more randomness. For each coordinate function, in the first phase, the 16 cofactors are calculated, refreshed, and registered using inputs from the first share domain. In the second phase, the inputs from the second share domain serve as selection bits to choose the correct cofactor out of the sixteen for output. In GHPC, the number of cofactors, registers, and randomness required is determined by the number of inputs and not the algebraic degree of the function.

3 Time Sharing Masking

We introduce our novel approach to securely first-order mask any (vectorial) Boolean function in hardware with a single register layer. We will refer to it using the acronym TSM, short for Time Sharing Masking, in the remainder of the paper. We begin the explanation with a toy example, applying TSM to a single AND gate, in Section 3.1. In Section 3.2, we write out TSM formally so it applies to any Boolean function, in particular also vectorial Boolean functions. Finally, in Section 3.3, we prove that TSM is first-order glitch-extended PINI composable secure.

3.1 Preliminary Example

Let x and y be the two inputs of the AND gate that computes $z = x \cdot y$. And let $(x_0, x_1), (y_0, y_1)$ be their sharings such that $x = x_0 + x_1$ and $y = y_0 + y_1$. A key aspect of TSM is to separate *in time* the processing of $share_0$ inputs from the processing of $share_1$ inputs with the help of a register layer, see Figure 1. Before the computation begins, we refresh the inputs with two random bits r_3, r_4 for the masked AND gate to be composable secure (see Section 3.3):

$$\begin{aligned} x'_0 &= x_0 + r_3 & \qquad x'_1 &= x_1 + r_3 \\ y'_0 &= y_0 + r_4 & \qquad y'_1 &= y_1 + r_4 \end{aligned}$$

Then, in the first phase, all cross product combinations of $share_0$, *i.e.*, $(x'_0, y'_0, x'_0 y'_0)$, are computed, refreshed, and stored in registers. In the second phase, all cross product combinations of $share_1$, *i.e.*, $(x'_1, y'_1, x'_1 y'_1)$, are computed. Finally, in order to produce the output of the AND gate, products of masked combinations of $share_0$ and combinations of $share_1$ are summed up, see Eq. (1).

$$\begin{aligned} z_0 &= \Big\{x'_0 y'_0 + r_0\Big\} + \Big\{x'_0 + r_1\Big\} \cdot \Big\{y'_1\Big\} + \Big\{y'_0 + r_2\Big\} \cdot \Big\{x'_1\Big\} \\ z_1 &= \Big\{r_0\Big\} + \Big\{r_1\Big\} \cdot \Big\{y'_1\Big\} + \Big\{r_2\Big\} \cdot \Big\{x'_1\Big\} + \Big\{x'_1\Big\} \cdot \Big\{y'_1\Big\} \end{aligned} \tag{1}$$

The computation requires five fresh random bits and eight registers to store the intermediate shares. The area and randomness utilization for computing a single AND gate is high, but we use this toy example only for illustration. TSM should be mainly applied to mask more complex non-linear functions as a whole, and not individual AND gates. The advantages of this approach will become more apparent in the remainder of the paper.

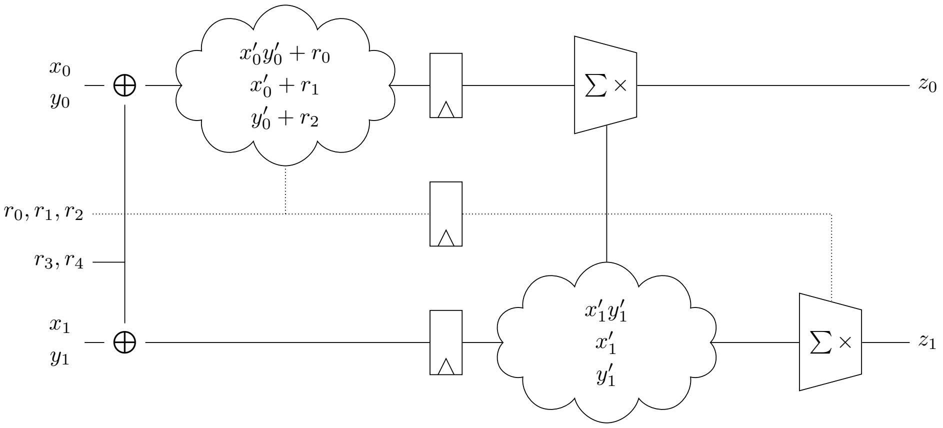


Figure 1: Application of TSM to a single AND gate.

3.2 Formal Description

We provide a description of TSM working on an arbitrary (vectorial) Boolean function. The outline is presented in Figure 2.

Specifically in this section, we change the notation to denote bits in a word by square brackets $(x[0], ..., x[k-1])$ instead of using different letters (*e.g.*, x, y in the previous section). We denote $x \in \mathbb{F}_2^k$ a k-bit word where its two-share Boolean masking is denoted by $\bar{x} = (x_0, x_1) \in \mathbb{F}_2^{2k}$ such that $x_0 + x_1 = x$ with $x_0 = (x_0[0], ..., x_0[k-1])$ and $x_1 = (x_1[0], ..., x_1[k-1])$ the notation of the share-words in separate bits. This change allows a simpler, more compact presentation of what follows next.

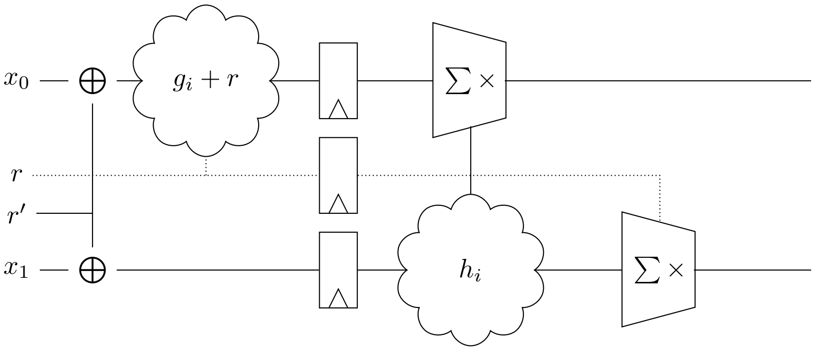


Figure 2: Application of TSM to an arbitrary (vectorial) Boolean function described by the functions g_i and h_i.

We explain the TSM method in a constructive manner where we first rewrite in Eq. (2) the algebraic normal form of a shared function as the sum of non-complete terms where each term is the multiplication between share domain 0 and share domain 1. We then rewrite this equation to Eq. (3) by adding fresh randomness allowing us to safely form the two output shares outlined in Eq. (4). Finally, the inputs of the gadget are first re-masked to ensure composable security.

We start informally, where we first rewrite the equations of a shared monomial. Namely, note that for the product of the bits $x[j]$ for some set of indices $j \in J \subset \{0, ..., k-1\}$

$$\prod_{j \in J} x[j] = \prod_{j \in J} (x_0[j] + x_1[j]) = \sum_{i \in \mathbb{F}_2^k} \prod_{j \in J} x_{i[j]}[j] = \sum_{i \in \mathbb{F}_2^k} \prod_{j \in J \text{ s.t. } i[j]=0} x_0[j] \prod_{j \in J \text{ s.t. } i[j]=1} x_1[j]\,.$$

In words, each monomial can be split as the sum of non-complete terms and each of these terms can be split as the multiplication of shares from domain 0 and shares from domain 1.

Consider an arbitrary Boolean function

$$f : \mathbb{F}_2^k \to \mathbb{F}_2 : x = (x[0], ..., x[k-1]) \mapsto f(x[0], .., x[k-1])\,.$$

We denote its two-share masking by $\bar{F} : \mathbb{F}_2^{2k} \to \mathbb{F}_2^2 : \bar{x} \mapsto (F_0(\bar{x}), F_1(\bar{x}))$ such that $F_0(\bar{x}) + F_1(\bar{x}) = f(x_0 + x_1)$. The above insight can be applied to each monomial in the algebraic normal form of f. We thus say that there exist functions g_i and h_i such that

$$f(x_0 + x_1) = \sum_{I \in \mathcal{P}_k} g_{\pi(I)}((x_0[i])_{i \in I}) h_{\pi(I)}((x_1[i])_{i \in \Omega/I})\,, \tag{2}$$

where we denote $(x_0[i])_{i \in I}$ as the set of all bits $x_0[i]$ for i in I. We also denote by $\mathcal{P}_k$ the power set of the indices $\Omega = \{0, ..., k-1\}$, namely all possible sets of indices in Ω. It is clear that $|\mathcal{P}_k| = 2^k$. The sets in $\mathcal{P}_k$ are numbered and indicated by the function π.

The functions g_i, h_i in Eq. (2) work only on share domain 0 and 1, respectively. To go back to the masked AND gate example from Section 3.1, the functions g_i, h_i are the following

$$\begin{array}{cccc} g_0(x_0, y_0) = x_0 y_0 & g_1(x_0) = x_0 & g_2(y_0) = y_0 & g_3(\emptyset) = 1 \\ h_0(\emptyset) = 1 & h_1(y_1) = y_1 & h_2(x_1) = x_1 & h_3(x_1, y_1) = x_1 y_1\,. \end{array}$$

Multiplying and adding the above terms, for $\mathcal{P}_2 = ((0,1), (0), (1), \emptyset)$, we get

$$\begin{aligned} xy &= \sum_{I \in \mathcal{P}_2} g_{\pi(I)}((x_0[i])_{i \in I}) h_{\pi(I)}((x_1[i])_{i \in \Omega/I}) \\ &= g_0(x_0, y_0) h_0(\emptyset) + g_1(x_0) h_1(y_1) + g_2(y_0) h_2(x_1) + g_3(\emptyset) h_3(x_1, y_1) \\ &= x_0 y_0 + x_0 y_1 + x_1 y_0 + x_1 y_1\,. \end{aligned}$$

The above sharing is already correct, however, it misses randomness for its security.

We thus further adapt Eq. (2) by adding randomness. Namely, by adding 2^k random bits $r = (r_0, ..., r_{2^k - 1})$, we get

$$\begin{aligned} f(x_0 + x_1) = &\sum_{I \in \mathcal{P}_k} (g_{\pi(I)}((x_0[i])_{i \in I}) + r_{\pi(I)}) h_{\pi(I)}((x_1[i])_{i \in \Omega/I}) \\ &+ \sum_{I \in \mathcal{P}_k} r_{\pi(I)} h_{\pi(I)}((x_1[i])_{i \in \Omega/I})\,. \end{aligned} \tag{3}$$

By re-masking, we can split the computation in two parts (read two phases), the computation and refreshing of $g_i(\cdot)$ on the first shares, and the computation and recombination of $h_i(\cdot)$ on the second shares. This is also depicted in Figure 2. In this figure, we also observe that the shares x_0, x_1 are first refreshed with the randomness $r' \in \mathbb{F}_2^k$. This is done in order to make TSM composable secure as proven in Section 3.3.

Finally, the two shares $F_0(\bar{x}), F_1(\bar{x})$ are composed as follows

$$\begin{aligned} F_0(\bar{x}) &= \sum_{I \in \mathcal{P}_k} (g_{\pi(I)}((x_0[i])_{i \in I}) + r_{\pi(I)}) h_{\pi(I)}((x_1[i])_{i \in \Omega/I}) \\ F_1(\bar{x}) &= \sum_{I \in \mathcal{P}_k} r_{\pi(I)} h_{\pi(I)}((x_1[i])_{i \in \Omega/I})\,. \end{aligned} \tag{4}$$

Since the functions g_i and h_i (or their product) consist of all terms up to degree k, any Boolean function can be made from these g_i and h_i. This is extended for vectorial functions ($\mathbb{F}_2^k \to \mathbb{F}_2^\ell$) by re-using the g_i and h_i functions for each coordinate function.

While this can make the output shares non-uniform (in the extreme case, two coordinate functions are equal), the security comes from the register layer being filled with uniquely re-masked values and from each function working only on one share domain at a time. This is made formal in the next section where we show that TSM is PINI composable secure.

3.3 Security

We prove that TSM is first-order probing secure and that it is, moreover, composable first-order secure in the Probe-Isolating Non-Interference (PINI) framework by Cassiers *et al.* [CS20]. Namely, we show that any circuit secured with TSM allows for trivial composition. Since TSM is designed to work over hardware, we use the glitch-extended probing model by Faust *et al.* [FGP$^+$18] to extend the PINI framework into the glitch-extended PINI framework. This PINI security is particularly important since it allows for the composition between gadgets without the need to place additional registers between them. Since all maskings of linear layers (where the linear function is applied share-wise) are PINI secure, we can trivially secure linear functions without adding additional registers or additional randomness.

Before starting the proof that the approach delivers PINI secure solutions, we need to introduce the necessary concepts to introduce PINI security. We start by providing the notion of simulation.

Definition 1 (Simulatability [CS20])**.** Let $P = \{p_1, ..., p_\ell\}$ be a set of ℓ probes of a gadget C and C_P the tuple of values of the probes for an execution of C. Let $I = \{(i_1, j_1), ..., (i_k, j_k)\} \subset \{0, ..., d-1\} \times \{0, ..., m-1\}$ be a set of input wires of C. A simulator is a randomized function $\mathcal{S} : \mathbb{F}_q^k \to \mathbb{F}_q^\ell$. The set of probes P can be simulated with the set of input wires I if there exists a simulator $\mathcal{S}$ such that for any inputs $x_{*,*}$, the distributions $C_P(x_{*,*})$ and $\mathcal{S}(x_{i_1,j_1}, ..., x_{i_k,j_k})$ are equal, where the probability is over the random coins in C and $\mathcal{S}$.

The above definition defines the security game in terms of a simulation game. This framework is extended to PINI security where we define which information is given to the simulator.

Definition 2 (PINI [CS20])**.** Let G be a gadget over d shares and P a set of t_0 (glitch-extended) probes on wires of G (called internal probes). Let A be a set of t_1 share indices. G is t-PINI if for all P and A such that $t_0 + t_1 \le t$, there exists a set B of at most t_1 share indices such that probes on the set of wires $P \cup y_{A,*}$ can be simulated with the wires $x_{A \cup B}$, with $x_{i,*}$ denoting all inputs of share i and $y_{i,*}$ denoting all outputs of share i.

Given the above definition of PINI, we show that any circuit secured by the TSM method from Section 3 is composable secure. Intuitively, the reason TSM is composable secure is due to each registered value (in the single register stage of the method) being re-masked by unique randomness.

Theorem 1. *Any circuit secured by TSM (Section 3) is first-order glitch-extended PINI.*

Proof. Denoting the k-bit input shares x_i and the output shares y_i. Looking at Definition 2 for $t = 1$ (considering glitch-extended probes), we find that we need to prove two cases. Namely,

- for $t_0 = 0$ and $t_1 = 1$, in which case we need to show that the output shares y_i can be simulated using the input shares x_i for $i \in \{0, 1\}$.
- for $t_0 = 1$ and $t_1 = 0$, in which case we need to show that a single intermediate probe can be simulated using either x_0 or x_1.

We begin with the first case, we have to prove that y_i can be simulated using x_i. We split up the proof depending on i.

- For y_0, the output is calculated from the values g_i re-masked by r and by values h_i which operate on the second shares x_1 re-masked by r'. Since g_i is re-masked by r and x_1 is re-masked by r', a simulator can sample r and r' and perfectly simulate the values y_0 as uniform randomness (in particular, the simulator does not need the values x_0).
- For y_1, since it is created using only x_1 and randomness r', a simulator can perfectly simulate y_1 given x_1 and by uniformly random sampling r' (in fact, due to r' the simulation would also work from scratch in which case the simulator can simulate the probed values as uniform randomness).

For the proof of the second case where we simulate an intermediate probe, we consider only probes in the first phase of the circuit, since probes on the second phase were already considered in the previous case. However, for probes on the first phase, it is clear that these can be perfectly simulated since the computation is done share-wise (a probe either only sees values from x_0 or from x_1) in which case the simulator simply gets either the zero or the one shares and performs the computation following the algorithm.

Since both cases are proven, the masking is first-order glitch-extended PINI. □

As a result, since the TSM circuit is first-order glitch-extended PINI secure, it can be composed with any other PINI gadget (which includes all linear operations too) without adding extra register stages or randomness following the proofs of composable security from the original work [CS20].

4 Advantages of TSM

In this section we mainly outline the general efficiency of TSM and contrast it with the first-order case of GLM and GHPC_{LL}. In general, we emphasize the advantages of TSM through a comparison of area, considering both the number of registers and logic, as well as the randomness required for masking a $\mathbb{F}_2^k \to \mathbb{F}_2^k$ function with an algebraic degree of $k-1$ (which represents the highest algebraic degree for a k-bit permutation).

4.1 Registers and Randomness Cost

In Section 3.2, we described our approach by applying it to an arbitrary Boolean function $f : \mathbb{F}_2^k \to \mathbb{F}_2$. The Boolean function is computed as a combination of the functions g_i and h_i. Importantly, the functions g_i and h_i solely depend on the k shared inputs. We emphasize that TSM can be extended to vectorial Boolean functions with multiple outputs ($\mathbb{F}_2^k \to \mathbb{F}_2^k$) because all coordinate functions share the same k inputs. The intermediate registers which store the refreshed results of the g_i functions, the random bits, and the second share inputs can be commonly used to calculate the shared outputs of all coordinate functions without increasing the register and randomness cost. In other words, the number of intermediate registers and randomness remains constant irrespective of the number of outputs.

TSM requires at most $2^k - 2$ registers to store the results of the g_i functions since $|\mathcal{P}_k| = 2^k$ and there is no degree k term (removing one register) and we do not store a constant term (removing the second register). TSM then requires at most another $2^k - 2$ registers to store the random bits r. Finally, TSM requires k registers to store the second share inputs. This gives a total of at most $2^{k+1} + k - 4$ registers. Similarly, for the randomness, TSM requires at most $2^k - 2$ bits to refresh the g_i functions and another k bits for r' in Figure 2.

We compare these numbers with GLM and GHPC_{LL} in Table 1. We note that both TSM and GLM can be more efficient than what is reported in the table, depending on the function to which the method is applied. Namely, we report the *worst case metrics* such that any function of degree $k-1$ can be implemented with the given register and randomness costs.

Table 1: Comparison for a $\mathbb{F}_2^k \to \mathbb{F}_2^k$ function of algebraic degree $k-1$.

Name	# Registers	# Register Layers	# Random Bits
TSM	$2^{k+1}+k-4$	1	2^k+k-2
GLM [GIB18]	$2^k k$	1	$2^k k$
GHPC_{LL} [KSM22]	$2^k k+k$	1	$2^k k$

We observe roughly a factor $k/2$ improvement in the number of registers and a factor k in random bits over both GLM and GHPC_{LL}. As previously mentioned, this improvement is a direct result of re-using the registered values for each coordinate function (of the k outputs).

If we compare TSM with GLM for masking an AES S-Box, we see significant, concrete savings in registers and random bits when implementing a higher algebraic degree function with many outputs. Groß *et al.* [GIB18] report the cost of masking an AES S-Box with a single register layer to be $16 \cdot 2^7 (= 2048)$ registers and $16 \cdot 2^7 (= 2048)$ random bits. To compare these numbers with TSM, we fill in the value $k = 8$ in Table 1. TSM requires only 516 registers and only 262 random bits per AES S-Box.

In Section 6, we discuss the implementation of the AES S-Box with TSM in more detail and provide concrete numbers for the area cost, including combinational logic.

4.2 Combinational logic

Without loss of generality, let us consider the PRINCE S-box, a 4×4 function $S : \mathbb{F}_2^4 \to \mathbb{F}_2^4$ for illustration. We show the equations for the function S in Algebraic Normal Form (ANF) in Eq. (5). We denote (a, b, c, d) as the four input bits and f^0, f^1, f^2, f^3 as the coordinate functions which produce the four output bits.

$$\begin{aligned}
f^0 &= 1 + dc + b + cb + dcb + a + da + ba \\
f^1 &= 1 + db + cb + dcb + ca + cba \\
f^2 &= d + dc + a + da + ca + dca + cba \\
f^3 &= 1 + c + cb + dcb + a + dca + ba + dba
\end{aligned} \tag{5}$$

Before delving into the benefit of TSM, let us briefly discuss how the function S would be masked using GLM. In the first stage, every coordinate f^0, f^1, f^2, f^3 is split into eight share domains. Cubic terms, such as bcd, are split into eight shared multiplication terms, $b_0c_0d_0, b_0c_0d_1, ..., b_1c_1d_1$. One multiplication term is assigned to each of the eight shared domains of the coordinate functions. Quadratic terms, such as bc, are split into four shared multiplication terms $b_0c_0, b_0c_1, b_1c_0, b_1c_1$ and are distributed among four of the eight shared domains of the coordinate functions. The share domains are then refreshed with fresh randomness and are registered. In the second stage, share compression is performed to reduce the number of shares from eight to two. In summary, the first stage involves expanding the number of shares, followed by the second stage, where the shares are compressed. On the other hand, to mask the function S, GHPC applies "Shannon decomposition", an identity that splits any Boolean function into parts called cofactors, to each coordinate function f^0, f^1, f^2, f^3. The coordinate functions are independently expanded into 16 cofactors by setting one share of the inputs a,b,c, and d to either 0 or 1.

A common characteristic between GLM and GHPC, which may be regarded as a potential drawback, is that every coordinate function is treated as a separate entity even though they commonly share the same inputs.

Applying TSM to the function S, in the first stage, all inputs are remasked, then all cross-products of the $share_0$ inputs are computed, *i.e.*, $(a_0, b_0, a_0b_0, ..., b_0c_0d_0)$, and finally those are refreshed and stored in the register layer. In the second stage, the cross-products of the $share_1$ inputs are computed, and they are then multiplied and summed with the masked cross-products of the $share_0$ inputs to produce the outputs of the coordinate functions.

TSM allows to reduce the cost of combinatorial logic by efficient reuse in several ways. First, we can deduplicate identical terms across coordinate functions, i.e. compute them only once and then reuse them. For example, dc is needed to compute f_0 and f_2 but there is no need to compute dc twice. Overall this allows to reduce the number of distinct terms to compute from 20 to 14. The decrease in logic becomes more prominent with an increase of the number of coordinate functions. This also reduces the number of random bits needed for refreshing in phase 1, and the number of registers.

Second, we can reuse already computed lower degree terms to compute higher degree terms. For example, we can compute dc and reuse it for computing dcb. Eq. (6) shows the sharing of the coordinate function f^0 and Eq. 7 shows in the first three lines the straightforward computation for $(b)_0$, $(cb)_0$ and $(dcb)_0$. In the fourth line it shows a more efficient computation of $(dcb)_0$ by reusing the already computed $(cb)_0$ which results in a reduction of the number of logic gates, thereby lowering the area. The decrease in logic gates becomes more prominent with an increase in the algebraic degree of the terms, particularly when masking higher algebraic degree functions such as the AES S-Box.

$$\begin{aligned} f_0^0 &= 1 + (dc)_0 + (b)_0 + (cb)_0 + (dcb)_0 + (a)_0 + (da)_0 + (ba)_0 \\ f_1^0 &= (dc)_1 + (b)_1 + (cb)_1 + (dcb)_1 + (a)_1 + (da)_1 + (ba)_1 \end{aligned} \tag{6}$$

$$\begin{aligned} (b)_0 &= \left\{b'_0 + r_1\right\} \\ (bc)_0 &= \left\{b'_0c'_0 + r_7\right\} + \left\{c'_0 + r_2\right\} \cdot \left\{b'_1\right\} + \left\{b'_0 + r_1\right\} \cdot \left\{c'_1\right\} \\ (bcd)_0 &= \left\{b'_0c'_0d'_0 + r_{13}\right\} + \left\{b'_0c'_0 + r_7\right\} \cdot \left\{d'_1\right\} + \left\{b'_0d'_0 + r_8\right\} \cdot \left\{c'_1\right\} + \left\{c'_0d'_0 + r_9\right\} \cdot \left\{b'_1\right\} \\ &\quad + \left\{b'_0 + r_1\right\} \cdot \left\{c'_1\right\} \cdot \left\{d'_1\right\} + \left\{d'_0 + r_3\right\} \cdot \left\{b'_1\right\} \cdot \left\{c'_1\right\} + \left\{c'_0 + r_2\right\} \cdot \left\{d'_1\right\} \cdot \left\{b'_1\right\} \\ (bcd)_0 &= (bc)_0 \cdot \left\{d'_1\right\} + \left\{b'_0c'_0d'_0 + r_{13}\right\} + \left\{b'_0d'_0 + r_8\right\} \cdot \left\{c'_1\right\} + \left\{c'_0d'_0 + r_9\right\} \cdot \left\{b'_1\right\} \\ &\quad + \left\{d'_0 + r_3\right\} \cdot \left\{b'_1\right\} \cdot \left\{c'_1\right\} \end{aligned} \tag{7}$$

4.2.1 Optimization during Logic Synthesis

Importantly, since the combinational logic in phase 1 (before the register layer) and the combinational logic in phase 2 (after the register layer) is non-complete, it is safe to allow (or even, one should enforce) the logic optimization through Electronic Design Automation (EDA) tools, without the need to carefully place logic in distinct modules. The only kind of optimization which must not be allowed is register re-timing, as that may move combinational logic across the register stage which may lead to a violation of non-completeness. Our case studies in the following sections include the impact of logic optimization on area, maximum frequency and security.

5 Case Study: Application of TSM to PRINCE

For our first case study, we chose to implement a masked version of the PRINCE block cipher [BCG+12] with TSM as a proof of concept. PRINCE is one of the few ciphers designed primarily for low-latency applications. With the rise of embedded devices with real-time requirements such as fast encryption or decryption, PRINCE fills the void of fast, lightweight block ciphers.

PRINCE is a 64-bit block cipher with a 128-bit key and consists of 12 rounds. The key is split into two 64-bit keys. One half is used to produce whitening keys (k_0, k_0') and the other half (k_1) is used in $PRINCE_{core}$ for the round key addition, see Figure 3. Each round of PRINCE consists of a key addition, a substitution layer (S, S^{-1}), a linear layer $(SR, SR^{-1}, M, M', M^{-1})$ and the addition of a round constant $(RC_0, RC_1, \ldots, RC_{11})$. For more details about the cipher, we refer the reader to the original paper [BCG+12].

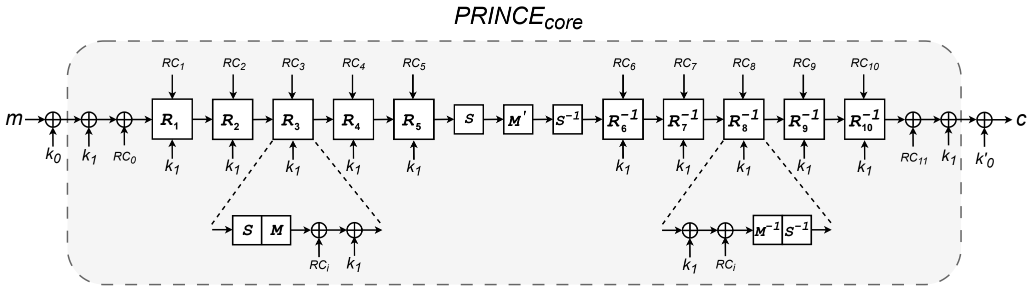

PRINCE_core
RC1 RC2 RC3 RC4 RC5 RC6 RC7 RC8 RC9 RC10
m
R1 R2 R3 R4 R5 S M' S^-1 R6^-1 R7^-1 R8^-1 R9^-1 R10^-1
c
k0 k1 RC0 k1 k1 k1 k1 k1 k1 k1 k1 k1 k1 RC11 k1 k'0
S M RCi k1
k1 RCi M^-1 S^-1

Figure 3: Schematic view of 12 rounds of PRINCE. Regular rounds R_i consist of an S-Box layer (S) and a linear layer M, and inverse rounds R_i^{-1} consist of an inverse S-Box layer (S^{-1}) and an inverse linear layer M^{-1}.

5.1 Masked PRINCE S-Box

The difficulty of masking a cipher implementation typically lies with its non-linear elements. The non-linear substitution layer (S)/inverse substitution layer (S^{-1}) of PRINCE consists of 4-bit S-Boxes of maximum algebraic degree three. We have already introduced the PRINCE S-box in Section 4. Our masked S-Box with a latency of one cycle is shown in Figure 4. Adhering to the guidelines for efficient implementation outlined in Section 4, we re-use the common degree-2 and degree-3 terms across coordinate functions, thus reducing the number of terms to compute, refresh and store from 20 to 14. Additionally, we capitalize on the shared outputs of degree-2 terms to calculate degree-3 terms.

Masking a single PRINCE S-Box with TSM requires 18 fresh random bits (4 for initial remasking and 14 for refreshing) and 32 intermediate registers $(14 + 14 + 4)$. In Table 2, we compare the utilization results of our masked PRINCE S-Box with other relevant PINI secure low-latency designs from the literature.

The synthesis results are gathered using the NanGate 45nm Open Cell Library [NAN]. We use Synopsys DC Compiler v2021.06 for Synthesis and provide results for two different sets of options. The first set of options (`compile -exact_map`) aims for a direct mapping of our design to logic. The second set of options (`compile_ultra -no_autoungroup -no_boundary_optimization`) aims for maximum logic optimization while making sure that no logic is moved across the register layer, see Section 4.2.1.

Although the `compile_ultra` option should not be carelessly used for masked implementations, because typically careful placement of logic in distinct modules is required, it can be safely done with TSM, as long as register retiming is not permitted. Later in Section 5.4 we demonstrate that TSM indeed remains secure with either set of these options, without careful placement of logic in distinct modules, and show that it becomes insecure if register retiming is enabled. The area numbers were gathered by synthesizing our designs with a target frequency of 100 MHz.

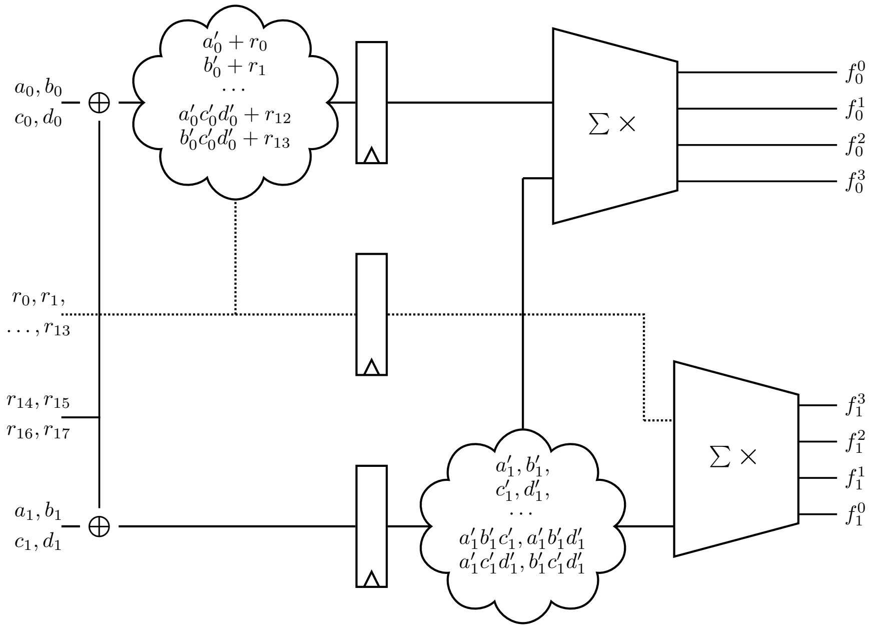


Figure 4: Architecture of our masked PRINCE S-Box.

Our single-cycle masked PRINCE S-Box has both lower area as well as randomness cost when compared to the GHPC$_{\text{LL}}$ design proposed by Knichel *et al.* [KM22]. The other designs need more than one single cycle. The exact sharings of our PRINCE S-Box are given in the HDL source code[1].

Table 2: Utilization results of relevant first-order masked PINI secure PRINCE S-Boxes.

Design	Method	Area (GE)	Rand (bits)	Cycles
This work	**TSM**	**538**[a] **453**[b]	**18**	**1**
[KSM22]	GHPC$_{\text{LL}}$	987	64	1
[KSM22]	GHPC$_{\text{LL}}$-AND	445	24	2
[KSM22]	GHPC	1384	4	2

[a] `compile -exact_map`
[b] `compile_ultra -no_autoungroup -no_boundary_optimization`

5.2 Round-Based Architecture

As this work aims to achieve low latency, the substitution layer consisting of 16 S-Boxes is implemented in parallel. We looked at the existing literature for efficient round-based architectures for PRINCE and chose to implement a first-order secure version of the unprotected architecture presented by Moradi *et al.* [MS16]. The PRINCE S-Box and its inverse (S^{-1}) are affine equivalents. By carefully constructing the architecture for the cipher, it is possible to minimize the additional circuitry needed to implement the inverse substitution layer. We can use affine transformations and re-use existing S-Box circuitry to compute S^{-1}. The inverse S-Box can be computed by applying input and output transformations to the S-Box as follows: $S^{-1} = A \circ S \circ A$. The input and output transformations applied to the S-Box are identical, A: `5764FDCE1320B98A`.

[1] https://github.com/KULeuven-COSIC/TSM

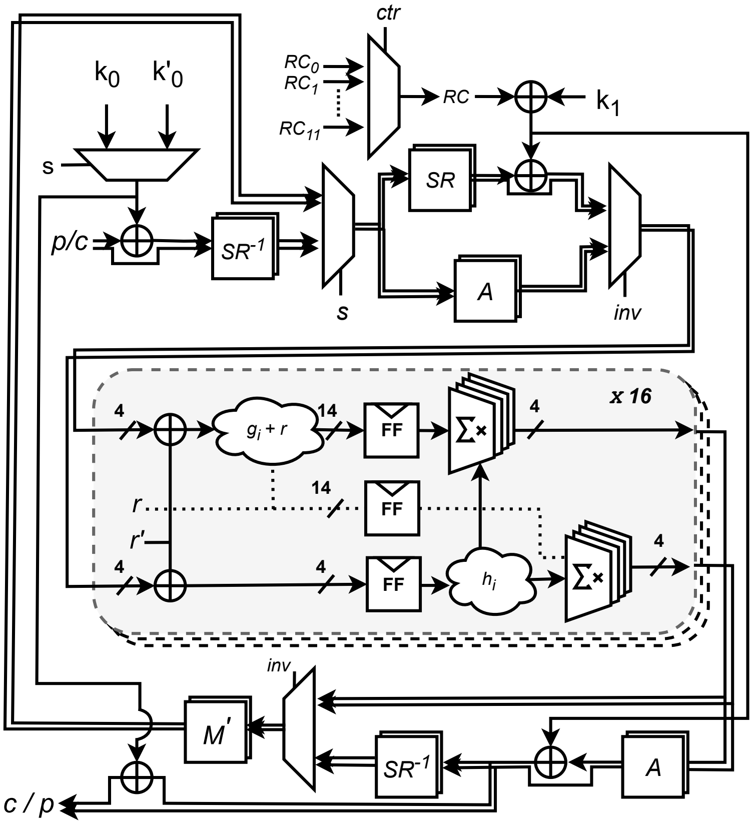


Figure 5: Architecture of our masked PRINCE design.

Our construction for the first-order secure PRINCE implementation using TSM is shown in Figure 5. The optimized single-cycle design requires only one substitution layer and uses the input/output transformation to calculate the inverse substitution layer. This reduces the area of our implementation. The matrix M' is an involution, and the other matrices M and M^{-1} used in the $PRINCE_{core}$ can be derived from M' through nibble shuffling operations SR and SR^{-1}, *i.e.* $M = SR \circ M'$ and $M^{-1} = M' \circ SR^{-1}$. Hence, we avoid implementing three matrices to save some area. All linear operations are duplicated for the two shares. Our key and round constants are not shared and are applied to only one of the two shares. Overall sixteen copies of the shared S-Box, see Figure 4, are implemented for parallel execution. The register stage is integrated within the shared S-Box. There are no additional state registers outside the S-Box, making our implementation's latency one cycle per round and thus 12 cycles for PRINCE.

5.3 Efficiency and Comparison

The utilization results of our implementation are summarized in Table 3. The maximum frequency was obtained by synthesizing our designs while iteratively increasing the target frequency until the slack became negative. We have further included all first-order secure PRINCE designs with a low latency of one cycle per round available in the literature, to the best of our knowledge.

From the area results, we can see that our design has the smallest area footprint by a good margin. With the `compile` option, it is about 32% smaller than the GLM design presented by Muller *et al.* [MMM21], which had the smallest area footprint among all existing low-latency designs before this work. With the `compile_ultra` option, the area becomes about 44% smaller than GLM. The other low-latency designs ([AZN21], [BNR19], [MMM21]) are all based on Threshold Implementations (TI). As TI relies on non-completeness for its security, all these designs process data which are split into more than two shares. The increased share count results in a larger area. The second smallest design in Table 3, from [MMM21], is based on GLM and requires implementing both the substitution and the inverse substitution layer. The security of GLM relies upon the condition that there are no variable collisions, *i.e.*, all the inputs must be independently shared. We refer the reader to the original GLM paper by Groß *et al.* [GIB18], where they discuss variable collisions and why it is necessary for the inputs to the S-Box to have independent sharings. The authors of [MMM21] argue that the affine transformations, required to compute the inverse S-Box, cause some of the inputs of the S-Box to be dependent and to violate non-completeness. This limitation of GLM implies that both the regular substitution layer consisting of S and its inverse consisting of S^{-1} must be implemented to avoid collisions. As a result, the additional circuit for S^{-1} increases their design area. However, in contrast, TSM does not require independent sharings of the inputs, because of the initial remasking with r' in Figure 2. With our method, affine transformations can be used to calculate S^{-1}. This advantage helps us achieve the lowest area compared to all other designs, with a one cycle per round latency. We note that like in the designs from Muller *et al.* [MMM21], which we mainly compare our design to, our design also does not mask the key. This ensures our area comparisons to be fair. The table also highlights the positive impact of logic optimization on the maximum clock frequency of our design.

Table 3: Utilization results of relevant first-order masked PRINCE implementations.

Design	Method	Area (GE)	f_{max} (Mhz)	Rand (bpc)*	Cycles
This Work	**TSM**	**13685**[a]	**485**[a]	**288**	**12**
		10926[b]	**610**[b]		
[MMM21]	GLM	20046	-	128	12
[AZN21]	LLTI	25857	488	48	12
[MMM21]	4-share TI	26158	-	0	12
[BNR19]#	TI	41628	335	48	12
[MMM21]	5-share TI	42158	-	0	12

[a] `compile -exact_map`

[b] `compile_ultra -no_autoungroup -no_boundary_optimization`

* bits per cycle

# Resynthesized with the NANGATE45 standard cell library by the authors of [AZN21]

In terms of randomness usage, our design uses more bits per cycle compared to other designs. However, we remind the reader of the results from Section 4.1. For masking any arbitrary function, *i.e.*, when comparing worst case metrics, TSM consumes less randomness than GLM. But we would like to note that the efficiency can be better depending on the concrete function that is masked. In the paper by Muller *et al.* [MMM21], their GLM based design consumes 256 random bits per cycle, which is comparable to our design's requirement of 288 bits per cycle. However, to reduce the costs, they re-use randomness over S-Boxes, which reduces it to 128 bits per cycle. Such optimizations come at the cost of losing the composable security of the general approach. Since composable security was one of the goals of this work, we leave further optimizations specifically for PRINCE, as well as randomness optimizations of a PINI secure general approach, as future work.

Our area estimation does not include the area overhead of PRNGs required to generate randomness, but this is consistent with the literature in general and the other designs included in Table 3 in particular. We use AES-128 in Output Feedback (OFB) mode to generate the randomness, which might be inefficient for the high randomness demands of low-latency applications. But we see random bit generation as a closely related yet orthogonal problem, and did not try to optimize this part. In the recent work by Cassiers *et al.* [CMM+23], the authors investigate several PRNGs and determine unrolled implementations of stream ciphers Trivium [Can06] and Bivium [Rad06] to be the best candidates for efficiently generating random bits. To provide an estimate of the PRNG area overhead, generating 288 random bits per cycle for our design would require approximately 6.5 kGEs and 10.2 kGEs using unrolled implementations of Bivium and Trivium, respectively.

5.4 Formal Verification

In addition to the general security proof for TSM presented in Section 3.3, we validate the security of our TSM PRINCE S-box using the formal verification tool SILVER [KSM20]. The tool verifies whether a masked implementation is secure and composable under various security notions such as probing security, Non-Interference (NI), Strong Non-Interference (SNI), and Probe-Isolating Non-Interference (PINI). We synthesized our PRINCE S-Box Verilog code to generate netlists with both the `compile` and `compile_ultra` options we discussed in Section 5.3. As shown in Table 4, the generated netlists for both options pass all security tests, which guarantees that our PRINCE S-Box is indeed probing and composable secure.

In order to increase confidence, we generate and verify an additional netlist of our S-Box after intentionally enabling register retiming, which is a performance optimization technique that allows combinational elements of our S-Box to be moved forward or backward across registers. The security of our masking technique relies on separating computations on shares with a register layer. Any movement of combinational logic across registers would very likely violate non-completeness. We set the `set_optimize_registers` condition to `true` and generate a netlist with the `compile_ultra -retime` option. The resulting netlist fails all robust probing tests. We conclude that our S-Box is secure when compiled with `compile_ultra` as long as register retiming remains disabled.

Table 4: Verification of our PRINCE S-Box with SILVER using different compiler options.

Compiler Option	Probing		NI		SNI		PINI		Uni.
	std.	rob.	std.	rob.	std.	rob.	std.	rob.	
`compile -exact_map`	✓	✓	✓	✓	✓	✓	✓	✓	✓

`compile_ultra -no_autoungroup`
`-no_boundary_optimization`
	✓	✓	✓	✓	✓	✓	✓	✓	✓

`set_optimize_registers true`
`compile_ultra -retime`
	✓	✗	✓	✗	✓	✗	✓	✗	✓

6 Case Study: Application of TSM to the AES S-Box

For our second case study, we examine the AES S-Box, a larger function with a higher algebraic degree, to demonstrate the efficiency of TSM. As outlined in Section 4, the main benefit of TSM comes from the ability to consolidate the coordinate functions of an S-Box and reuse the intermediate registered values among them. As one would expect, for larger S-Boxes with many coordinate functions, such as AES, the benefits of TSM should be more evident compared to other low-latency masking techniques that independently mask each coordinate function with unique sharings. In this section, we use TSM to create a low-latency AES S-Box and compare our implementation costs with other relevant low-latency masking techniques.

An unprotected AES S-Box processes eight input bits and generates eight output bits, denoted as $S : \mathbb{F}_2^8 \rightarrow \mathbb{F}_2^8$. We represent the eight input bits as (a, b, c, d, e, f, g, h) and the eight outputs using individual coordinate functions, each denoted as $f : \mathbb{F}_2^8 \rightarrow \mathbb{F}_2$. The coordinate functions written in their Algebraic Normal Form (ANF) include all terms up to degree 7. In total, there are 254 terms, calculated as the sum of binomial coefficients for each degree from 1 to 7: $\sum_{i=1}^{7} \binom{8}{i} = 254$.

6.1 Masked AES S-Box

At the high level, our strategy for masking the AES S-Box closely resembles the general TSM approach. In the initial phase, we calculate all cross-products of the $share_0$ inputs up to degree 7, *i.e.* excluding the degree 8 term $a_0b_0c_0d_0e_0f_0g_0h_0$, and refresh them with 254 bits of randomness. In the following phase, we calculate the cross-products of the $share_1$ inputs and combine them with the masked cross-products of the $share_0$ inputs to produce the S-Box outputs. When applying TSM to a large S-Box such as AES, the register cost forms only a small fraction of the total area. The majority of the total area is attributed to the logic computation of coordinate functions in the second phase. Consequently, optimizing logic becomes a crucial factor. We therefore apply the ideas explained in Section 4.2, and in particular extend them to all higher degree (degree-4,-5,-6 and -7) terms. An example with the sharings for the degree-4 term $abcd$ is shown in Eq. (8). We use the already computed degree-3 and degree-2 terms: acd, bcd, cd to construct the sharing of $abcd$. By following a similar strategy, for degree-5 terms, we use the outputs of the degree-4 and degree-3 terms. For degree-6 terms, we use the outputs of the degree-5, degree-4, and degree-3 terms. Finally, for degree-7 terms, we use the outputs of the degree-6, degree-5, and degree-4 terms. For additional details, we provide our HDL source code[2] with the precise sharings of our complete AES S-Box.

$$\begin{array}{l}
(abcd)_0 = (acd)_0 \cdot \left\{b'_1\right\} + (bcd)_0 \cdot \left\{a'_1\right\} + (cd)_0 \cdot \left\{a'_1\right\} \cdot \left\{b'_1\right\} + \left\{a'_0b'_0c'_0d'_0 + r_{14}\right\} \\
\qquad + \left\{a'_0b'_0c'_0 + r_{10}\right\} \cdot \left\{d'_1\right\} + \left\{a'_0b'_0d'_0 + r_{11}\right\} \cdot \left\{c'_1\right\} + \left\{a'_0b'_0 + r_4\right\} \cdot \left\{c'_1\right\} \cdot \left\{d'_1\right\} \\
\hline
(abcd)_1 = (acd)_1 \cdot \left\{b'_1\right\} + (bcd)_1 \cdot \left\{a'_1\right\} + (cd)_1 \cdot \left\{a'_1\right\} \cdot \left\{b'_1\right\} + \left\{r_{14}\right\} + \left\{r_{10}\right\} \cdot \left\{d'_1\right\} \\
\qquad + \left\{r_{11}\right\} \cdot \left\{c'_1\right\} + \left\{r_4\right\} \cdot \left\{c'_1\right\} \cdot \left\{d'_1\right\}
\end{array} \tag{8}$$

6.2 Efficiency and Comparison

We summarize the utilization results for our masked S-Box in Table 5 and compare them with other state-of-the-art low-latency masking schemes. We synthesized our design with the same compiler options and switches we used for PRINCE in Section 5.1. All designs were synthesized with a target clock frequency of 100 MHz to get the area results. To find the maximum frequency of our designs, we follow the same strategy of gradually increasing the frequency until we observe a negative slack.

With `compile`, our AES S-Box has an area of 20.5 kGEs, which is approximately one-third of the area of GLM and GHPC_{LL}. We attribute the lower area cost to the two aspects of TSM we discussed in Section 4. First, the sharings for the 254 terms are commonly reused among all 8 coordinate functions. Second, reusing already computed lower degree terms for computing higher degree terms substantially reduces the area for logic. Using the `compile_ultra` compiler option further reduces the logic area, resulting in a much smaller design with just below 14.5 kGEs. Also here we note the positive impact of logic optimization on the maximum clock frequency. Still our S-Box is larger

[2] https://github.com/KULeuven-COSIC/TSM

than the S-Box in LMDPL. But it is noteworthy that the AES S-Box in LMDPL is based on dual-rail logic, which requires a pre-charge phase between consecutive evaluations. A comparison of a single AES S-box in TSM and in LMDPL is thus not straightforward.

Table 5: Utilization results of relevant first-order masked AES S-Box implementations.

Design	Method	Area	f_{max}	Rand	Latency
		(GE)	(MHz)	(bits)	(Cycles)
This Work	**TSM**	**20523**[a]	**375**[a]	**262**	**1**
		14352[b]	**535**[b]		
[SBHM20]	LMDPL	3480	400	36	1
[GIB18]	GLM	60730	356	2048	1
[KSM22]	GHPC_{LL}	64111	-	2048	1
[LMW14]	LMDPL	2830	-	36	2
[GIB18]	GLM	6740	584	416	2
[AZN21]	4-share LLTI	25780	277	0	2
[AZN21]	4-share TI	58410	40	0	2
[KSM22]	GHPC	77145	-	8	2

[a] `compile -exact_map`

[b] `compile_ultra -no_autoungroup -no_boundary_optimization`

Comparing randomness usage, our S-Box requires roughly eight times less randomness when compared to both GLM and GHPC_{LL}. The reduction can be directly linked to the consolidation of coordinate functions and the reuse of the 254 shared terms. Given that the AES S-Box produces eight output bits, we observe an eight-fold decrease. LMDPL requires less randomness but again a comparison of a single S-Box is difficult.

As for the maximum frequency, it is not an easy comparison as most designs use a different standard cell library. Our design has a maximum frequency similar to GLM and LMDPL.

6.3 Formal Verification

We were not able to formally verify the netlists of our AES S-Box, primarily due to the limitations of SILVER, which uses Binary Decision Diagrams (BDD) to represent Boolean functions and generate probability distributions for verification and statistical independence checks. Since our masked AES S-Box is significantly larger than the masked PRINCE S-Box, the tool terminates with a message indicating that the unique table storing BDD nodes has been filled. As an alternative to formal verification, we experimentally assess the leakage of our AES S-Box on an FPGA, see the next section.

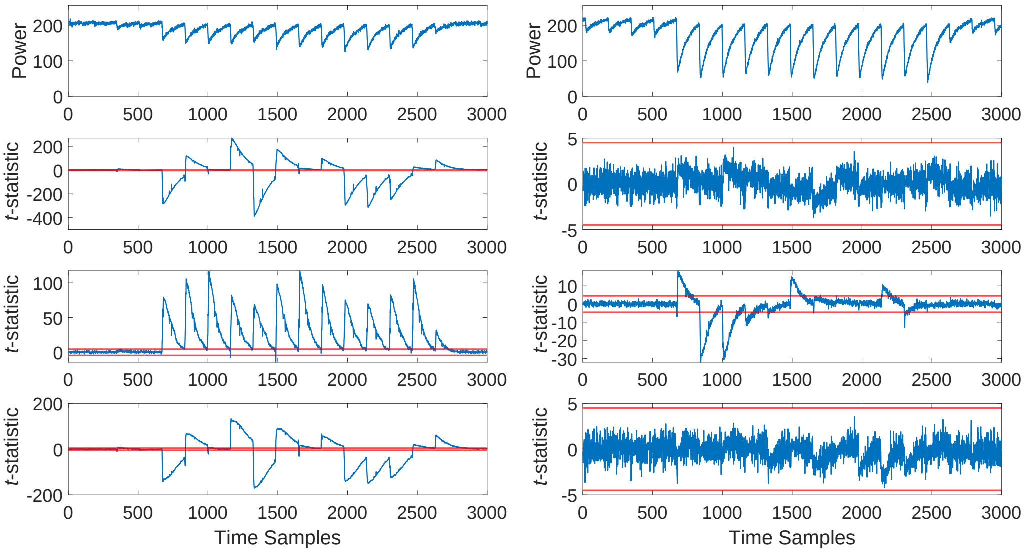
7 Practical Evaluation

We evaluate our first-order protected designs by implementing and checking for side-channel leakage on FPGA. We use a SAKURA-G board [Pro13], featuring two Xilinx SPARTAN-6 FPGAs, and Xilinx ISE v14.7. As commonly done, we implement our protected design (complete PRINCE or AES S-Box) on the target FPGA where it receives pre-shared inputs from the control FPGA and returns shared outputs to the control FPGA in order to avoid any IO leakage. We use the control FPGA to interface with a PC and to orchestrate the sequence of inputs to the target FPGA. We use an (unprotected) AES-128 encryption core in Output Feedback (OFB) mode as a PRNG on the control FPGA. In case of the complete PRINCE design, the target FPGA contains a wrapper module that implements PRNGs to generate online randomness. The seed for these PRNGs is generated on the control

FPGA and transferred to the target FPGA for every PRINCE encryption/decryption. The protected PRINCE core receives random bits from the PRNGs in the wrapper as needed in every clock cycle.

We measure the (on-board amplified) power consumption of the target FPGA on SMA connector J3 with a Tektronix DPO7254C oscilloscope sampling at 500MS/s. We operate the board at a very slow clock frequency of 3 MHz to ensure that there is no overlap between the power patterns of consecutive clock cycles. The clock signal is stable and provided by an external function generator synchronized with the oscilloscope to enable synchronous sampling. This is done to ensure the best possible alignment of the traces to reduce sampling noise. The power measurements cover the entire 12 rounds for the protected PRINCE, and one S-Box computation for AES. The top rows of Figures 6 and 7 show sample power traces (raw oscilloscope ADC output).

We use the non-specific Test Vector Leakage Assessment (TVLA) methodology as explained by Bilgin *et al.* [BGN+14] and originally described in [GJJR11, CDMG+13]. Concretely we use the fast computation method described by Reparaz *et al.* [RGV17]. We focus on univariate analysis since we claim no first-order leakage. We perform fixed vs. random tests where we provide either a fixed or a random input to the masked PRINCE core resp. to the masked AES S-Box, in a random sequence. The unprotected keys are fixed for all experiments.

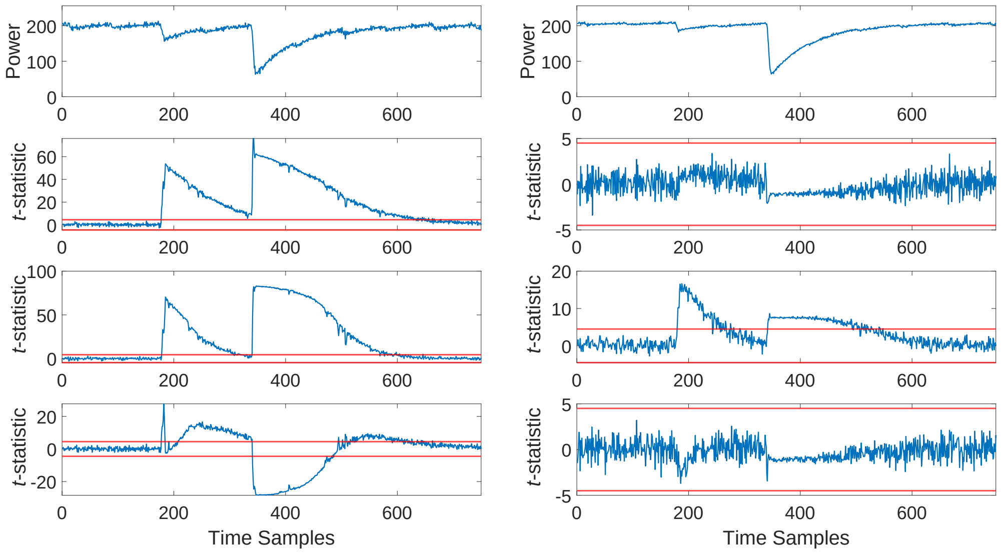


(a) PRNGs set to *OFF* with 100k traces. **(b)** PRNGs set to *ON* with 100 million traces.

Figure 6: TVLA results for masked PRINCE core. Top to bottom: power trace, the first-, second-, and third-order univariate t-test result. The red lines indicate the ±4.5 threshold.

As a safety check, we first provide a result of the leakage assessment performed with the PRNGs switched off, which means all random bits used by the masked PRINCE core, by the masked AES S-Box, and for initial masking, are zero. Figure 6a and Figure 7a show for 100k traces, from top to bottom: the first-order univariate t-test result, second-order univariate t-test result, and third-order univariate t-test result. Red lines indicate the commonly applied threshold at ±4.5. We obtain very significant peaks in the first-order t-test, which confirms that our setup works well.

Figure 6b and Figure 7b show the TVLA results with the PRNGs switched ON. We measured and analyzed 100 million traces of the complete PRINCE core and 250 million traces of the AES S-Box. As we can see from the first-order t-test results in the second



(a) PRNGs set to *OFF* with 100k traces. **(b)** PRNGs set to *ON* with 250 million traces.

Figure 7: TVLA results for AES S-Box. Top to bottom: power trace, the first-, second-, and third-order univariate t-test result. The red lines indicate the ±4.5 threshold.

rows, the t-test value does not exceed the threshold of ±4.5 at any time sample, therefore, indicating that our designs do not exhibit first-order leakage. The processing of a non-linear function must combine information from all shares at some point. TSM uses two shares and thus we expect to observe second-order leakage, which is indeed evident in the second-order t-test results. We show this result to increase confidence in the correctness of our measurement and processing. One can further expect to observe third-order leakage, because the data in TSM's register layer is essentially in three shares. However, we do not observe leakage in the third-order t-test results, one possible explanation is that more traces would be required.

8 Conclusion

We introduced a novel approach for low-latency masking that ensures security by separating the processing of the different shares over time with a register layer. TSM is straightforward to apply to any arbitrary (vectorial) Boolean function and does not come with constraints that a designer must be careful about to avoid compromising security. We prove first-order glitch-extended PINI security and demonstrate excellent implementation results in terms of low overheads when implementing a complete PRINCE core and an AES S-box.

More specifically, the masking method scales in register and randomness cost on the number of inputs and the algebraic degree of the function, but it amortizes this cost for each output bit. The overhead reduction of TSM comes from the ability to consolidate masked component functions and use the intermediate values commonly among them.

We demonstrated the effectiveness of our approach by implementing a first-order secure low-latency PRINCE that has 32% resp. 44% area improvement over the state-of-the-art. We formally validated the security of our PRINCE S-Box implementation with SILVER, and we provide results of practical leakage assessment on the complete PRINCE core which show no first-order leakage with 100 million traces.

Our masked AES S-box costs roughly one-third resp. one-quarter of the area of state-of-the-art implementations, and has much lower randomness cost as well. We deliberately exclude LMDPL from the comparison because a direct comparison to TSM seems difficult. The AES S-Box in LMDPL is smaller and requires less random bits, at first sight, but it requires dual-rail logic and a precharge phase between consecutive executions, which will become important when building a circuit which executes over more than one clock cycle. We provide results of practical leakage assessment of our AES S-Box which show no first-order leakage with 250 million traces.

As a future work, it would be interesting to reduce the randomness cost of TSM without trading off latency and security. Using an additional register layer (on the output), it is possible to re-use the random bits r over all gadgets. For our PRINCE implementation, this would mean 14 random bits can be re-used among all 16 S-Boxes. However, without such an additional register layer, the reduction of randomness seems less trivial. Another direction for future work is to extend and generalize TSM to higher-order security. It would also be interesting to implement more algorithms with TSM in order to compare the overhead reductions with other existing low-latency designs.

Acknowledgment.

This research has been funded in part by KU Leuven Startfinanciering STG/20/047, by CyberSecurity Research Flanders VR20192203, by the European Commission through Belfort ERC Advanced Grant 101020005 695305, and through Horizon Europe RIA grant HORIZON-CL3-2021-CS-01-02 101070008 ORSHIN.

References

[AZN21] Victor Arribas, Zhenda Zhang, and Svetla Nikova. LLTI: low-latency threshold implementations. *IEEE Trans. Inf. Forensics Secur.*, 16:5108–5123, 2021.

[BCG+12] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. PRINCE - A low-latency block cipher for pervasive computing applications (full version). *IACR Cryptol. ePrint Arch.*, page 529, 2012.

[BGN+14] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Higher-order threshold implementations. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II*, volume 8874 of *Lecture Notes in Computer Science*, pages 326–343. Springer, 2014.

[BNR19] Dusan Bozilov, Ventzislav Nikov, and Vincent Rijmen. Design trade-offs in threshold implementations. In *26th IEEE International Conference on Electronics, Circuits and Systems, ICECS 2019, Genoa, Italy, November 27-29, 2019*, pages 751–754. IEEE, 2019.

[Can06] Christophe De Cannière. Trivium: A stream cipher construction inspired by block cipher design principles. In Sokratis K. Katsikas, Javier López, Michael Backes, Stefanos Gritzalis, and Bart Preneel, editors, *Information Security, 9th International Conference, ISC 2006, Samos Island, Greece, August 30 - September 2, 2006, Proceedings*, volume 4176 of *Lecture Notes in Computer Science*, pages 171–186. Springer, 2006.

[CDMG$^+$13] J. Cooper, E. De Mulder, G. Goodwill, J. Jaffe, G. Kenworthy, and P. Rohatgi. Test vector leakage assessment (TVLA) methodology in practice. http://icmc-2013.org/wp/wp-content/uploads/2013/09/goodwillkenworthtestvector.pdf, 2013.

[CGLS21] Gaëtan Cassiers, Benjamin Grégoire, Itamar Levi, and François-Xavier Standaert. Hardware private circuits: From trivial composition to full verification. *IEEE Trans. Computers*, 70(10):1677–1690, 2021.

[CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.

[CMM$^+$23] Gaëtan Cassiers, Loïc Masure, Charles Momin, Thorben Moos, Amir Moradi, and François-Xavier Standaert. Randomness generation for secure hardware masking - unrolled trivium to the rescue. Cryptology ePrint Archive, Paper 2023/1134, 2023. https://eprint.iacr.org/2023/1134.

[CS20] Gaëtan Cassiers and François-Xavier Standaert. Trivially and efficiently composing masked gadgets with probe isolating non-interference. *IEEE Trans. Inf. Forensics Secur.*, 15:2542–2555, 2020.

[FGP$^+$18] Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. Composable masking schemes in the presence of physical defaults & the robust probing model. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):89–120, 2018.

[GIB18] Hannes Groß, Rinat Iusupov, and Roderick Bloem. Generic low-latency masking in hardware. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(2):1–21, 2018.

[GJJR11] G. Goodwill, B. Jun, J. Jaffe, and P. Rohatgi. A testing methodology for side channel resistance validation. http://csrc.nist.gov/newsevents/non-invasive-attack-testing-workshop/08Goodwill.pdf, 2011.

[GMK16] Hannes Groß, Stefan Mangard, and Thomas Korak. Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. In Begül Bilgin, Svetla Nikova, and Vincent Rijmen, editors, *Proceedings of the ACM Workshop on Theory of Implementation Security, TIS@CCS 2016 Vienna, Austria, October, 2016*, page 3. ACM, 2016.

[GMO01] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic analysis: Concrete results. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, volume 2162 of *Lecture Notes in Computer Science*, pages 251–261. Springer, 2001.

[GP99] Louis Goubin and Jacques Patarin. DES and differential power analysis (the "duplication" method). In Çetin Kaya Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99, Worcester, MA, USA, August 12-13, 1999, Proceedings*, volume 1717 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 1999.

[ISW03] Yuval Ishai, Amit Sahai, and David A. Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[KM22] David Knichel and Amir Moradi. Low-latency hardware private circuits. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 1799–1812. ACM, 2022.

[Koc96] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

[KSM20] David Knichel, Pascal Sasdrich, and Amir Moradi. SILVER - statistical independence and leakage verification. In Shiho Moriai and Huaxiong Wang, editors, *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*, volume 12491 of *Lecture Notes in Computer Science*, pages 787–816. Springer, 2020.

[KSM22] David Knichel, Pascal Sasdrich, and Amir Moradi. Generic hardware private circuits towards automated generation of composable secure gadgets. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(1):323–344, 2022.

[LMW14] Andrew J. Leiserson, Mark E. Marson, and Megan A. Wachs. Gate-level masking under a path-based leakage metric. In Lejla Batina and Matthew Robshaw, editors, *Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings*, volume 8731 of *Lecture Notes in Computer Science*, pages 580–597. Springer, 2014.

[MMM21] Nicolai Müller, Thorben Moos, and Amir Moradi. Low-latency hardware masking of PRINCE. In Shivam Bhasin and Fabrizio De Santis, editors, *Constructive Side-Channel Analysis and Secure Design - 12th International Workshop, COSADE 2021, Lugano, Switzerland, October 25-27, 2021, Proceedings*, volume 12910 of *Lecture Notes in Computer Science*, pages 148–167. Springer, 2021.

[MPG05] Stefan Mangard, Thomas Popp, and Berndt M. Gammel. Side-channel leakage of masked CMOS gates. In Alfred Menezes, editor, *Topics in Cryptology - CT-RSA 2005, The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005, Proceedings*, volume 3376 of *Lecture Notes in Computer Science*, pages 351–365. Springer, 2005.

[MS16] Amir Moradi and Tobias Schneider. Side-channel analysis protection and low-latency in action - - case study of PRINCE and midori -. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 517–547, 2016.

[NAN] NANGATE. The nangate 45nm open cell library. https://www.nangate.com.

[NGPM22] Rishub Nagpal, Barbara Gigerl, Robert Primas, and Stefan Mangard. Riding the waves towards generic single-cycle masking in hardware. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(4):693–717, 2022.

[NRR06] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold implementations against side-channel attacks and glitches. In Peng Ning, Sihan Qing, and Ninghui Li, editors, *Information and Communications Security, 8th International Conference, ICICS 2006, Raleigh, NC, USA, December 4-7, 2006, Proceedings*, volume 4307 of *Lecture Notes in Computer Science*, pages 529–545. Springer, 2006.

[Pro13] SAKURA Hardware Security Project. Sakura-G: Side-channel AttacK User Reference Architecture. https://satoh.cs.uec.ac.jp/SAKURA/hardware/SAKURA-G.html, 2013.

[QS01] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (EMA): measures and counter-measures for smart cards. In Isabelle Attali and Thomas P. Jensen, editors, *Smart Card Programming and Security, International Conference on Research in Smart Cards, E-smart 2001, Cannes, France, September 19-21, 2001, Proceedings*, volume 2140 of *Lecture Notes in Computer Science*, pages 200–210. Springer, 2001.

[Rad06] Havard Raddum. Cryptanalytic results on trivium. estream, ecrypt stream cipher project, report 2006/039, 2006, 2006.

[RBN$^+$15] Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating masking schemes. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015*, volume 9215 of *Lecture Notes in Computer Science*, pages 764–783. Springer, 2015.

[RGV17] Oscar Reparaz, Benedikt Gierlichs, and Ingrid Verbauwhede. Fast leakage assessment. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 387–399. Springer, 2017.

[SBHM20] Pascal Sasdrich, Begül Bilgin, Michael Hutter, and Mark E. Marson. Low-latency hardware masking with application to AES. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(2):300–326, 2020.