IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2023, No. 1, pp. 113–152. DOI:10.46586/tches.v2023.i1.113-152

Low-Latency and Low-Randomness Second-Order Masked Cubic Functions

Aein Rezaei Shahmirzadi[1],
Siemen Dhooghe[2] and Amir Moradi[1]

[1] Ruhr University Bochum, Horst Görtz Institute for IT Security, Bochum, Germany
firstname.lastname@rub.de
[2] imec-COSIC, ESAT, KU Leuven, Leuven, Belgium
firstname.lastname@esat.kuleuven.be

Abstract. Masking schemes are the most popular countermeasure to mitigate Side-Channel Analysis (SCA) attacks. Compared to software, their hardware implementations require certain considerations with respect to physical defaults, such as glitches. To counter this extended leakage effect, the technique known as Threshold Implementation (TI) has proven to be a reliable solution. However, its efficiency, namely the number of shares, is tied to the algebraic degree of the target function. As a result, the application of TI may lead to unaffordable implementation costs. This dependency is relaxed by the successor schemes where the minimum number of $d+1$ shares suffice for dth-order protection independent of the function's algebraic degree. By this, although the number of input shares is reduced, the implementation costs are not necessarily low due to their high demand for fresh randomness. It becomes even more challenging when a joint low-latency and low-randomness cost is desired. In this work, we provide a methodology to realize the second-order glitch-extended probing-secure implementation of cubic functions with three shares while allowing to reuse fresh randomness. This enables us to construct low-latency second-order secure implementations of several popular lightweight block ciphers, including Skinny, Midori, and Prince, with a very limited number of fresh masks. Notably, compared to state-of-the-art equivalent implementations, our designs lower the latency in terms of the number of clock cycles while keeping randomness costs low.

Keywords: Low Latency, Low Randomness, Masking, Side-Channel Analysis

1 Introduction

There have been numerous studies to investigate the security of embedded devices as an attacker can gain physical access to the target device. Hence, the attacker can gain execution related information by monitoring the device's power consumption and/or electromagnetic radiation and recover secret data. This implies that the underlying algorithm that are implemented on such devices should be mathematically secure while the implementation is also physically secure.

Seminal contributions have been made by Kocher et al. [KJJ99] by introducing Differential Power Analysis (DPA) in 1999, where the authors managed to recover the secret key of various algorithms by collecting power traces. This finding demonstrated the need for additional studies to understand more about its foundations and the development of countermeasures. Through several experimental analyses and scientific articles, it has been proven that masking is one of the best approaches to prevent side-channel analysis attacks. Based on secret-sharing schemes, masking techniques randomize the key-dependent intermediate values of the cipher. The most common approach in masking is Boolean masking

Received: 2022-07-15 Accepted: 2022-09-15 Published: 2022-11-29

in which the sensitive value is split into several shares whose XOR results in the original value. The application of Boolean masking on linear functions is straightforward, and the functions can be performed on each share individually. The more challenging part is the masked realization of non-linear functions where the design is opted for different criteria like latency, area, randomness, a specific platform or application, etc. in the open literature [GIB18, ZSS+21, KM22b, BDMS22, DSM22, KM22a, SBM21].

To evaluate the security of a given design, the probing model has been introduced in [ISW03]. In this model, the number of probes that the attacker can place in a circuit to observe intermediate values defines the order of the security. Although this model works well in software implementations where all operations are done sequentially, it has been shown that this model does not capture hardware implementations' leakage due to a phenomenon called glitches. Reparaz et al. [RBN+15] addressed the issue and introduced the *glitch-extended* probing model. A probe at the output of any gate is propagated backwards to either the last synchronization point (register) or the primary inputs leading up to the probed intermediate signal.

A Threshold Implementation (TI) [NRR06] is the first implementation strategy that is immune against glitches. Following this scheme and fulfilling its properties, a design can be implemented without any fresh masks during the execution of the protected cipher. While this techniques defines the number of input shares as $td+1$, where t is the algebraic degree of the function and d is the order of the security, other techniques have been proposed later that relaxed the requirements. The authors of [GMK16, RBN+15] demonstrated that $d+1$ input shares can be used when d-th order security is desired. Using these schemes can be beneficial when less area overhead is a requirement, however, it forces to use fresh randomness to maintain security. The generation of this randomness can be inefficient in hardware platforms. Furthermore, the cost of this generation is also not often reported in academic literature, which makes the comparison of some countermeasures unfair.

Recently, Daemen [Dae17] introduced a technique called the *changing of the guards* to avoid using fresh masks at every clock cycle to achieve uniformity. Later, Shahmirzadi and Moradi [SM20] presented a methodology to realize first-order secure hardware implementations with two shares which do not require fresh masks. While these techniques reduced the implementation costs in hardware platforms, it becomes more challenging to extend them to higher-order security.

Higher-order TI was first introduced by Bilgin et al. [BGN+14], however, it was quickly shown that this methodology is not secure due to the lack of fresh randomness leading to composability issues [Rep15]. A methodology has been presented in [SM21] resulting in efficient realization of a group of quadratic functions whose second-order secure hardware implementation requires no fresh masks. Nonetheless, apart from their Keccak sharing, 8-bit of fresh masks have been used per S-box to realize glitch-extended probing-secure implementations of popular symmetric primitives.

The authors of [BDZ20] proposed a framework to evaluate the security of higher-order threshold implementations enabling them to avoid fresh masks that need to be updated every clock cycles. They applied their approach to the LED cipher [GPPR11] to make a second-order secure design with some randomness that remains unchanged during the execution. However, the design has seven input shares which makes it inefficient on hardware platforms. Later, the framework was generalized in [BDMS22] and also the implementation costs have been improved. Nevertheless, they decomposed the non-linear layer into quadratic functions in their case studies forcing them to place registers between them. As a result, there is a need for second-order masked circuits that do not require a high amount of fresh randomness while maintaining latency.

1.1 Our Contributions

In this work, we provide masking techniques for three-share hardware constructions of cubic functions providing second-order security with a low fresh mask use and a low latency in terms of clock cycles. We start our study with cubic functions and show how to achieve glitch-extended probing security with a minimum number of input shares while using reusable fresh masks. We apply the methodology to the S-boxes of Midori, Skinny, and Prince, where each round is performed in only two clock cycles. We evaluate our S-box constructions by the leakage verification tool SILVER [KSM20] under the glitch-extended probing model and present a concrete probing security analysis for each case study that allows for the randomness in each masked S-box to be reused. For the sake of completeness, we verify the security of our implementations by FPGA-based practical experiments. The HDL representations of the constructed S-boxes and full ciphers are publicly available in GitHub.

2 Preliminaries

In this section, we introduce the probing model and its verification tools, Boolean masking, threshold implementations, and masking with $d + 1$ shares.

2.1 Notations

In this work, we operate over binary vector spaces $\mathbb{F}_2^n$. For the binary field $\mathbb{F}_2$, its field operations correspond to the XOR and AND gates. We denote operations over binary variables in its Algebraic Normal Form (ANF), for example $ab + c$ denotes the AND between the variables a and b and the XOR of its result with c.

We denote shares of binary variables with a subscript, *i.e.* the variable x_i denotes the ith share of x. The same holds for shared functions, where each share of the function is denoted by a subscript. For example, the coordinate f_i denotes the ith share of the function $\bar{F}$ and calculates the ith share of the function's output.

2.2 The Glitch-Extended Probing Model

Ever since the seminal work by Kocher et al. [KJJ99], there has been a considerable body of work on understanding the foundation of Side-Channel Analysis (SCA) and how to mitigate the attacks. Masking, as the most promising countermeasure, has been investigated by a great number of authors in literature leading to many different schemes [Tri03, ISW03, NRR06, RBN+15, GMK16, NRS11, GM18, GIB18]. An important question associated with the proposition of the masking schemes is how to model adversaries considering physical defaults and their different execution environments.

The probing model, first introduced by Ishai et al. [ISW03], is one of the first steps in the security assessment of masking schemes. In the dth-order probing model, an adversary is able to observe d intermediate wires of the circuit during the execution of the cipher. Security in this model is guaranteed by showing that the stochastic values returned by the probes can be simulated by a simulator which is not given the secret inputs to the circuit while the adversary and the real circuit are given this information. The security is achieved by using randomness generated by the simulator and the circuit which the adversary does not have access to. Practically speaking, probing security is proven by showing that the probed values follow a random distribution which is not affected by the choice of the secret value.

While the probing model is a good first step as a security model, hardware implementations can be insecure in practice while fulfilling its security requirements. The model provides security under the assumption that there is no data-dependent activation timing, which fits best on software platforms. This is due to a common phenomenon in CMOS technologies called glitches. The inaccurate assumption of the probing model not capturing the effect of glitches leads to insecure designs as shown in [MPO05, MME10]. More precisely, the physical characteristics such as transitions, coupling effects, or glitches are not considered in the d-probing model [FGP+18]. Hence, an extended model is needed for these unwanted effects.

Originally, Reparaz et al. [RBN+15] introduced an extension to the probing model to capture glitches. This was later put into the *robust-probing model* by Faust et al. [FGP+18]. This model covers the physical properties of hardware platforms. For example, to cover the effect of glitches, a *glitch-extended* probe is made where a probe on a combinatorial circuit is extended to all signals (up to registers or primary inputs) that involve in the computation of the probed wire. The introduction of such a simple model enabled the relevant scientific communities to develop formal verification tools to evaluate a small circuit [BBC+19, KSM20] (small due to the limitations of the complexity of the method). Furthermore, it helped to reduce the implementation cost of several schemes while maintaining the same level of security [BGR18, SM20, SM21].

2.3 Verification Tools

The development of automated formal verification tools for the probing model enables researchers to reduce the complexity of security proofs for masking schemes while also optimizing their implementations costs. Considering the recent tools dedicated to masked hardware designs, a language-based verification tool named MaskVerif is presented in [BBC+19]. This tool uses conservative heuristics leading to false positive cases. In other words, this tool may report the insecurity of a design that is actually secure. The authors of [KSM20] addressed this issue and presented a formal verification tool called SILVER. It works directly on the gate-level netlist of a hardware design and makes an exhaustive analysis of its probability distributions. It does not simplify anything and hence the results are reliable and the tool avoids false negatives. It reports the result of evaluations based on the security notions defined in [MBR19]. In this paper, we used SILVER to assess the second-order security of our designs under the "glitch-extended probing model". Since it can only handle small circuits and cannot deal with a full encryption module, we also confirmed the security of our constructions with a practical analysis on an FPGA evaluation board in Section 5.

2.4 Boolean Masking and Threshold Implementation

Boolean masking is a technique based on splitting each secret variable $x \in \mathbb{F}_2$ in the circuit into shares $\bar{x} = (x_1, x_2, \ldots, x_{s_x})$ such that $x = \sum_{i=1}^{s_x} x_i$ over $\mathbb{F}_2$. A random Boolean masking of a fixed secret is uniform if all sharings of that secret are equally likely.

In this paper, we will use the notions of threshold implementations as introduced by Nikova et al. [NRR06]. As such, we introduce the notions of non-completeness and uniformity.

A shared function $\bar{F}$ is non-complete if each of its coordinate functions f_i operate on all but one shares of each input secret. This notion has been extended by Bilgin et al. [BGN+14] to capture each set of d coordinate functions being jointly non-complete.

Definition 1 (dth-order non-complete)**.** A masked function $\bar{F}$ is d^{th}*-order non-complete* if any d coordinate functions f_i depend on at most $s_x - 1$ input shares.

The above notion was created as a necessary property to secure maskings against higher-order univariate attacks including the effect of glitches.

A shared function is called uniform if, when given a uniform input sharing, it outputs a sharing which is uniform.

Definition 2 (Uniformity [NRR06])**.** A shared function $\bar{F}(\bar{x}) = \bar{y}$ is uniform if $\forall x \in \mathbb{F}$, $\forall \bar{y} \in Sh(F(x))$:

$$\left| \left\{ \bar{x} \in Sh(x) \,\middle|\, \bar{F}(\bar{x}) = \bar{y} \right\} \right| = \frac{|\mathbb{F}|^{s_x - 1}}{|\mathbb{F}|^{s_y - 1}} ,$$

where $Sh(x)$ denotes the set of valid share vectors $\bar{x}$ of the secret x.

The notion of uniformity has been shown to help reduce randomness overheads in both first-order and higher-order designs. Special search algorithms are typically needed to find a non-complete and uniform sharing of a given function.

In this work, we will reduce randomness costs by employing threshold implementations in a higher-order setting. More specifically, considering that all shared functions are uniform, then in a second-order scenario multivariate security (if univariate security is already achieved) is ensured by having fresh masks be present after each possible probe position in that layer. The first probe's returned values are then re-masked causing the second probe to view independent data from the first probe. In case the masked function is first-order probing secure without the use of randomness, each probe's returned values separately need to be independent of the secret data. From this reasoning, we find that we only need to re-mask what can be observed by a probe and thus some randomness can be re-used when re-masking the S-boxes. For example, instead of re-masking the full state after an S-box calculation, it could suffice to re-mask the columns where the same randomness is used for each column. This technique was introduced as "resilient uniformity" by Dhooghe et al. [DN22] where the authors show a uniform second-order masked PRESENT where this trick is applied. However, the technique can also be viewed from the cryptanalytic framework by Beyne et al. [BDZ20], where the argument would be that no trails can exist between two probe positions due to the added randomness present in the design.

2.5 Masking with $d + 1$ Shares

It has been shown that the implementation cost of classical Threshold Implementations (TI) [NRR06] can be quite significant as the number of input shares increases based on the algebraic degree of the target function. It becomes more challenging to achieve higher-order security using TI [BGN+14] as it is insecure against multivariate adversaries [Rep15], and fresh masks should be added when composing the functions [MPL+11, CBR+15].

In order to use the minimum number of input shares, i.e., $d + 1$ shares for dth-order security, the authors of [GMK16] presented a methodology called Domain Oriented Masking (DOM) to achieve security in hardware platforms. However, it demands fresh randomness to achieve non-completeness even in a first-order secure design contrary to TI which can be secure without fresh masks. Following this technique, a two-share variant of a two-input AND gate $f(a, b) = x$ with a single-bit fresh mask r can be realized as follows

$$\begin{array}{llllr}
f_0(a_0, b_0) & = a_0 b_0 & \rightarrow x'_0 & & \\
f_1(a_0, b_1, r) & = a_0 b_1 + r & \rightarrow x'_1 & & x'_0 + x'_1 = x_0 \\
\hline
f_2(a_1, b_0, r) & = a_1 b_0 + r & \rightarrow x'_2 & & x'_2 + x'_3 = x_1 \\
f_3(a_1, b_1) & = a_1 b_1 & \rightarrow x'_3 & &
\end{array},$$

where a_0, a_1, b_0, b_1 are the input shares, and x_0, x_1 are the output shares. The coordinate functions' f_i result is stored in registers and its computation is referred to as the *expansion layer*. These outputs are then XORed to generate the output shares (this phase is known as the *compression layer*).

It has been shown that there is no need for fresh masks to achieve two-share first-order security for some quadratic functions [RBN+15]. This result has been extended by Shahmirzadi and Moradi [SM20] who presented an algorithm to realize a two-share first-order secure implementation of a cubic function without fresh masks. In another paper, the technique has been further extended to second-order security with three-shares in [SM21]. In this paper, we consider masking cubic S-boxes of several lightweight block ciphers decomposed into quadratic functions which require no fresh masks.

Nevertheless, even though we can create second-order maskings of cubic functions without fresh masks, some randomness should still be introduced at the intersection of the separate functions to assure multivariate security. These fresh masks should be updated each clock cycle and we refer to them as *dynamic* fresh masks. The goal of this paper is to keep the need for dynamic fresh masks at a minimum. Apart from dynamic fresh masks, we also use *static* masks. These are masks which are given to the design and can remain unchanged during the execution of the cipher.

Fixing the shares to $d+1$, there is a duality between latency and randomness. In order to reduce randomness, it is better to work with decomposed S-boxes. However, this inherently increases the latency and vice versa. Thus, there is a need for a technique which allows for both lower latency designs and for low randomness overheads.

3 Masking Techniques

In this section, we first explain our generic procedure allowing to find first-order glitch-extended probing secure constructions of 4-bit cubic functions without any fresh masks using three shares. Afterwards, we add several static masks that can be reused to realize a second-order secure implementation.

3.1 Finding a Uniform Sharing

As explained in Section 2.5, the general structure of a $d+1$ masked nonlinear operation is divided into two separated parts: the expansion layer and the compression layer. Since we intend to create second-order secure sharings, at least three input shares are used and the coordinate functions in the expansion layer should be second-order non-complete (Definition 1). In this paper, we focus on 4-bit cubic functions and hence we need a minimum of 27 coordinate functions to achieve second-order non-completeness. The minimum of 27 comes from the algebraic degree of the target function being three. As each input is represented by 3 shares, a three-shared cubic monomial is the sum of $3^3 = 27$ different cubic monomials of the shares. Each of these cubic monomials is assigned to a coordinate function to ensure second-order non-completeness.

Let us denote a cubic function by $x = f(a,b,c,d)$, where $\langle a,b,c,d\rangle$ are four bits. We refer to the coordinate functions and the output shares of the masked version by $x'_i = f'_i(.)$ and x_i, respectively. We represent the share index of the input variables given to the coordinate functions by a table called the *index configuration* as shown in Table 1 which bears some similarities to the one given in [ZSS+21]. Since in this paper we focus on 4-bit cubic S-boxes, we follow the sharing indices presented in by Bozilov et al. [BKN19] which guarantees a non-complete and correct sharing of any 4-bit cubic function. The table reflects the share index of each input variable that is involved in the calculation of each coordinate function. The coordinate functions, whose results are stored in a register layer, are categorized by dashed lines and are combined in the compression layer, i.e., $x_0 = \bigoplus_{i=0}^{8} x'_i$, $x_1 = \bigoplus_{i=9}^{17} x'_i$, $x_2 = \bigoplus_{i=18}^{26} x'_i$.

For example, if we assign $\langle I_0, I_1, I_2, I_3\rangle$ to $\langle a,b,c,d\rangle$, the first and second coordinate functions $f'_0()$ and $f'_1()$ receive $\{a_0, b_0, c_0, d_0\}$ and $\{a_0, b_0, c_1, d_1\}$ as their input list, respectively. To make it more clear in this example, we provided the full input list of coordinate

Table 1: Index configuration of our sharing.

#coordinate function	I_0	I_1	I_2	I_3
0	0	0	0	0
1	0	0	1	1
2	0	0	2	2
3	0	1	1	0
4	0	1	2	1
5	0	1	0	2
6	0	2	0	1
7	0	2	1	2
8	0	2	2	0
9	1	0	2	0
10	1	0	0	1
11	1	0	1	2
12	1	1	2	2
13	1	1	0	0
14	1	1	1	1
15	1	2	0	2
16	1	2	1	0
17	1	2	2	1
18	2	0	0	2
19	2	0	1	0
20	2	0	2	1
21	2	1	0	1
22	2	1	2	0
23	2	1	1	2
24	2	2	1	1
25	2	2	0	0
26	2	2	2	2

functions in Table 2. We should highlight that any combination of $I_{i\in\{0,1,2,3\}}$ can be assigned to the four input variables of the target function leading to $4! = 24$ different index configuration tables. Table 2 shows only one possibility where $\langle I_0, I_1, I_2, I_3\rangle$ is assigned to $\langle a, b, c, d\rangle$.

Once an index configuration table is assigned to the input variables, the place of all cubic monomials of the target function become clear. To demonstrate our approach, we assume that the target function is $f(a, b, c, d) = abc + bc + d$. For the cubic monomial abc, we have 27 shared cubic monomials in the masked variant, each of which can be placed only in one coordinate function. In other words, the ANF of each coordinate function can contain only one of the shared cubic monomials depending on the index configuration table. For instance, in the case shown in Table 2, the shared monomials $a_0b_0c_0$ and $a_0b_0c_1$ must be placed in coordinate functions $f'_0(a_0, b_0, c_0, d_0)$ and $f'_1(a_0, b_0, c_1, d_1)$, respectively. There is only one possibility for the rest shared cubic monomials as well. However, the place of the quadratic and linear terms are not fixed. We use this freedom to find a uniform sharing (Definition 2). In case of our exemplary target function, the quadratic monomial bc has four shared quadratic monomials, i.e., $b_0c_0, b_0c_1, b_1c_0, b_1c_1$. Contrary to shared cubic monomials, there are multiple possibilities for the placement of these shared monomials. For example, considering the index configuration shown in Table 2, the shared monomial b_0c_0 can be placed in $f'_0(.)$, $f'_{10}(.)$, and $f'_{18}(.)$. The same holds for linear terms where the shared linear monomials can be placed in different coordinate functions. As mentioned, we

Table 2: A sample index configuration tables, where $\langle I_0, I_1, I_2, I_3\rangle$ is assigned to $\langle a, b, c, d\rangle$.

#coordinate function	a	b	c	d	coordinate function
0	0	0	0	0	$f'_0(a_0, b_0, c_0, d_0)$
1	0	0	1	1	$f'_1(a_0, b_0, c_1, d_1)$
2	0	0	2	2	$f'_2(a_0, b_0, c_2, d_2)$
3	0	1	1	0	$f'_3(a_0, b_1, c_1, d_0)$
4	0	1	2	1	$f'_4(a_0, b_1, c_2, d_1)$
5	0	1	0	2	$f'_5(a_0, b_1, c_0, d_2)$
6	0	2	0	1	$f'_6(a_0, b_2, c_0, d_1)$
7	0	2	1	2	$f'_7(a_0, b_2, c_1, d_2)$
8	0	2	2	0	$f'_8(a_0, b_2, c_2, d_0)$
9	1	0	2	0	$f'_9(a_1, b_0, c_2, d_0)$
10	1	0	0	1	$f'_{10}(a_1, b_0, c_0, d_1)$
11	1	0	1	2	$f'_{11}(a_1, b_0, c_1, d_2)$
12	1	1	2	2	$f'_{12}(a_1, b_1, c_2, d_2)$
13	1	1	0	0	$f'_{13}(a_1, b_1, c_0, d_0)$
14	1	1	1	1	$f'_{14}(a_1, b_1, c_1, d_1)$
15	1	2	0	2	$f'_{15}(a_1, b_2, c_0, d_2)$
16	1	2	1	0	$f'_{16}(a_1, b_2, c_1, d_0)$
17	1	2	2	1	$f'_{17}(a_1, b_2, c_2, d_1)$
18	2	0	0	2	$f'_{18}(a_2, b_0, c_0, d_2)$
19	2	0	1	0	$f'_{19}(a_2, b_0, c_1, d_0)$
20	2	0	2	1	$f'_{20}(a_2, b_0, c_2, d_1)$
21	2	1	0	1	$f'_{21}(a_2, b_1, c_0, d_1)$
22	2	1	2	0	$f'_{22}(a_2, b_1, c_2, d_0)$
23	2	1	1	2	$f'_{23}(a_2, b_1, c_1, d_2)$
24	2	2	1	1	$f'_{24}(a_2, b_2, c_1, d_1)$
25	2	2	0	0	$f'_{25}(a_2, b_2, c_0, d_0)$
26	2	2	2	2	$f'_{26}(a_2, b_2, c_2, d_2)$

can form 24 different index configuration tables by assigning the different configurations I_i to the input variables. In the first step, we select one index configuration and make a correct sharing of the target function while maintaining second-order non-completeness in the coordinate functions. To do so, we just use one possible placement of the shared quadratic and linear terms to have a correct sharing. At this step, we only want to have a correct sharing where second-order non-completeness is fulfilled in the coordinate functions. We search for a uniform sharing in the next step.

To find a uniform sharing, we only consider linear terms as the search space would be too large when including quadratic terms. We refer to these terms that are added to achieve uniformity as correction terms following the work by Bilgin et al. [BNN+15]. Since we have already a correct sharing, we must add each correction term an even number of times to the output shares $\{x_0, x_1, x_2\}$ and since we are working with three shares this means that each correction term is added exactly twice. Hence for each correction term, denoted by z, we have four possibilities: 1) add it to first and second output shares$\{x_0 + z, x_1 + z, x_2\}$; 2) add it to first and third output shares$\{x_0 + z, x_1, x_2 + z\}$; 3) add it to second and third output shares$\{x_0, x_1 + z, x_2 + z\}$; 4) do not use it. Note that we cannot use the shares of the input variables assigned to I_0 as correction terms since only one share of the input value is involved in the computation of each output share (see Table 1). For example, in the case shown in Table 2, a_0, a_1, and a_2 cannot be used as correction terms.

As a result, since any correction term should be assigned to the coordinate functions that generate the output shares, it is not possible to add them to two different output shares without violating non-completeness. We can consider the other three input variables, each of which has three shares leading to nine different correction terms. As stated earlier, for each correction term we have four possibilities making the size of the search space equal to $4^9 = 2^{18}$ per index configuration table. As a result, we need to search through 24×2^{18} different possibilities for each cubic function, which is a large search space. For this reason, we do not search over quadratic terms for finding a uniform sharing. To check the uniformity of each solution, we go over all shared values of each unshared input. This way, we can discard one candidate earlier improving the run time of our program.

4-bit cubic bijections are commonly used for S-boxes in lightweight block ciphers, e.g., Midori [BBI+15], Skinny [BJK+16], Prince [BCG+12]. As a result, we focus our work on this class of functions. Each coordinate function of such a bijection can be at most cubic. Hence, we can apply the technique described above to each coordinate function. Fresh masks are not used in the construction of each shared coordinate function so they are not necessarily jointly uniform. Consequently, we should find a combination of coordinate functions that have this property. The number of solutions for each coordinate function is usually high leading to a significant number of combinations requiring an optimized search algorithm. We use a step-by-step approach where we first search for jointly-uniform solutions for two coordinate functions and then search for the third one that is jointly-uniform with the first two to finally find the last one. This technique has also been employed in several other publications [SM20, SM21] to optimize this search process.

3.2 Achieving First-Order Security

In this paper, we aim to fulfill the first-order security of the design without the use of randomness. We achieve this by pairing two S-boxes that make use of each other's inputs as shown in Figure 1. We note that this technique bears some similarity to the technique presented in [BDMS22]. However, we are dealing with cubic functions instead of quadratic ones. Since each coordinate function is second-order non-complete, the only place that adversaries can gain information about the secret is in the compression layer. More precisely, each probe extends to the output of nine different coordinate functions in the glitch-extended probing model and we require that these nine coordinate functions are jointly probing secure.

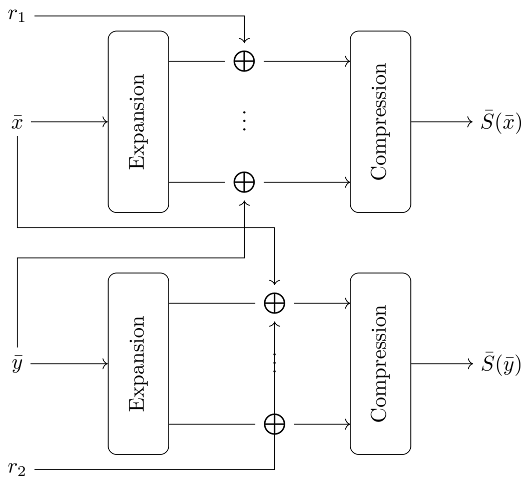


Figure 1: Depiction of the pairing of two masked S-boxes.

We demonstrate our approach by providing a uniform sharing of the function $x = f(a, b, c, d) = abc + bc + d$ using three shares which can be considered as one coordinate function of a 4-bit cubic S-box.

$$
\begin{array}{llll}
f_0(a_0,b_0,c_0,d_0) & = a_0b_0c_0 + b_0c_0 + d_0 + k_0 + r_0 + r_1 & \rightarrow x'_0 & \\
f_1(a_0,b_0,c_1) & = a_0b_0c_1 + b_0c_1 + k_0 + l_0 + r_1 + r_2 & \rightarrow x'_1 & \\
f_2(a_0,b_0,c_2) & = a_0b_0c_2 + b_0c_2 + l_0 + m_0 + r_2 + r_3 & \rightarrow x'_2 & \\
f_3(a_0,b_1,c_0) & = a_0b_1c_0 + m_0 + n_0 + r_3 + r_4 & \rightarrow x'_3 & \\
f_4(a_0,b_1,c_1) & = a_0b_1c_1 + n_0 + k_1 + r_4 + r_5 & \rightarrow x'_4 & \bigoplus_{i=0}^{8} x'_i = x_0 \\
f_5(a_0,b_1,c_2) & = a_0b_1c_2 + k_1 + l_1 + r_5 + r_6 & \rightarrow x'_5 & \\
f_6(a_0,b_2,c_0) & = a_0b_2c_0 + l_1 + m_1 + r_6 + r_7 & \rightarrow x'_6 & \\
f_7(a_0,b_2,c_1) & = a_0b_2c_1 + m_1 + n_1 + r_7 + r_8 & \rightarrow x'_7 & \\
f_8(a_0,b_2,c_2) & = a_0b_2c_2 + n_1 + r_8 + r_9 & \rightarrow x'_8 & \\
\hline
f_9(a_1,b_0,c_0,d_1) & = a_1b_0c_0 + d_1 + k_0 + r_9 + r_{10} & \rightarrow x'_9 & \\
f_{10}(a_1,b_0,c_1) & = a_1b_0c_1 + k_0 + l_0 + r_{10} + r_{11} & \rightarrow x'_{10} & \\
f_{11}(a_1,b_0,c_2) & = a_1b_0c_2 + l_0 + m_0 + r_{11} + r_{12} & \rightarrow x'_{11} & \\
f_{12}(a_1,b_1,c_0) & = a_1b_1c_0 + b_1c_0 + m_0 + n_0 + r_{12} + r_{13} & \rightarrow x'_{12} & \\
f_{13}(a_1,b_1,c_1) & = a_1b_1c_1 + b_1c_1 + n_0 + k_1 + r_{13} + r_{14} & \rightarrow x'_{13} & \bigoplus_{i=9}^{17} x'_i = x_1 \\
f_{14}(a_1,b_1,c_2) & = a_1b_1c_2 + b_1c_2 + k_1 + l_1 + r_{14} + r_{15} & \rightarrow x'_{14} & \\
f_{15}(a_1,b_2,c_0) & = a_1b_2c_0 + l_1 + m_1 + r_{15} + r_{16} & \rightarrow x'_{15} & \\
f_{16}(a_1,b_2,c_1) & = a_1b_2c_1 + m_1 + n_1 + r_{16} + r_{17} & \rightarrow x'_{16} & \\
f_{17}(a_1,b_2,c_2) & = a_1b_2c_2 + n_1 + r_{17} + r_{18} & \rightarrow x'_{17} & \\
\hline
f_{18}(a_2,b_0,c_0,d_2) & = a_2b_0c_0 + d_2 + k_0 + r_{18} + r_{19} & \rightarrow x'_{18} & \\
f_{19}(a_2,b_0,c_1) & = a_2b_0c_1 + k_0 + l_0 + r_{19} + r_{20} & \rightarrow x'_{19} & \\
f_{20}(a_2,b_0,c_2) & = a_2b_0c_2 + l_0 + m_0 + r_{20} + r_{21} & \rightarrow x'_{20} & \\
f_{21}(a_2,b_1,c_0) & = a_2b_1c_0 + m_0 + n_0 + r_{21} + r_{22} & \rightarrow x'_{21} & \\
f_{22}(a_2,b_1,c_1) & = a_2b_1c_1 + n_0 + k_1 + r_{22} + r_{23} & \rightarrow x'_{22} & \bigoplus_{i=18}^{26} x'_i = x_2 \\
f_{23}(a_2,b_1,c_2) & = a_2b_1c_2 + k_1 + l_1 + r_{23} + r_{24} & \rightarrow x'_{23} & \\
f_{24}(a_2,b_2,c_0) & = a_2b_2c_0 + b_2c_0 + l_1 + m_1 + r_{24} + r_{25} & \rightarrow x'_{24} & \\
f_{25}(a_2,b_2,c_1) & = a_2b_2c_1 + b_2c_1 + m_1 + n_1 + r_{25} + r_{26} & \rightarrow x'_{25} & \\
f_{26}(a_2,b_2,c_2) & = a_2b_2c_2 + b_2c_2 + n_1 + r_{26} + r_0 & \rightarrow x'_{26} & \\
\end{array}
$$

In black, we denote the uniform sharing of the function. To make the sharing first-order glitch-extended probing secure, we use the inputs of the paired S-box $\langle k, l, m, n\rangle$ which are denoted in green in the equations above. Obviously, we only use two out of three shares of each input variables and we should carefully add them in such a way that first-order non-completeness is fulfilled. These terms are added to all coordinate functions that are compressed to generate one output shares. As a result, regarding the first-order security with no fresh masks, probing an output share reveals the information about the paired S-box's input shares (at most two out of three shares). For instance, in the given example above with no fresh masks, probing the output share x_0 reveals $\langle k_0, l_0, m_0, n_0, k_1, l_1, m_1, n_1\rangle$. In other words, a probe on the output share x_0 in glitch-extended probing model is propagated backwards to the registered results of all coordinate functions x'_i for $0 \le i \le 8$. The probe on x'_8 reveals information about n_1, considering the fact that there are no fresh masks in the design yet. The probe on x'_7 reveals information about $m_1 + n_1$. Using the information about n_1 by the probe on x'_8, m_1 can be recovered. Following the probes one by one, the information on the other S-box's input $\langle k_0, l_0, m_0, n_0, k_1, l_1\rangle$ can be recovered as well. Since only two out of three shares can be recovered, the design is first-order secure with no fresh masks. We also confirmed the first-order security of our example using SILVER. The property that probing the compression layer of one S-box solely reveals inputs of the paired S-box will be used in the case studies in Section 4 to reduce the dynamic fresh randomness used to re-mask the output of the S-boxes. Essentially, in this technique, the paired S-box's input shares act as fresh randomness enabling us to achieve first-order security without added randomness cost. From this point of view, the technique bears some similarity with the changing of the guard technique [Dae17] in the sense that both use one masked S-box's input shares to mask another. However, the changing of the guards is a technique to achieve uniformity for a design fulfilling non-completeness property. Instead, we solve the non-completeness issue of a sharing which is uniform.

3.3 Extension to Second-Order Security

To achieve second-order security in the compression layer (as the expansion layer is already secure thanks to its second-order non-completeness), we add fresh randomness in a ring refreshing approach, as shown in red in the equations of the previous section. This addition of fresh randomness follows the methodology from Reparaz *et al.* [RBN+15] where it was shown to ensure the second-order probing security of the compression layer. For example, probing x_0 in the 3-input multiplier above would provide the values $y_0+r_0+r_1, ..., y_8+r_8+r_9$, with $y_0, ..., y_8$ values created from the inputs from the paired S-box. Each of these y_i is re-masked by unique random values providing no information to the adversary. The randomness for each S-box in a pair should be different. This ensures that probing the two S-boxes in a pair is still second-order secure. However, the randomness used in the S-box pair can be reused in other paired S-boxes. The reason is that probing two different pairs is still second-order secure even though their fresh masks are the same due to each S-box pair being first-order probing secure without fresh randomness. Thus, given that the two pairs work on independent inputs (for example, the two pairs are in the same S-box layer), then if adversaries can recover all fresh masks with one probe in one pair, the other pair is still first-order secure without these fresh masks and hence the second probe does not reveal information about the secret.

This leaves us with proving the inputs from all possible pairs of S-boxes are indeed independent. While this is already true for pairs in the same S-box layer (due to the joint uniformity of the state), we need to verify whether this holds between pairs in different rounds of the symmetric primitive. To ensure this independence, we use dynamic fresh randomness between the rounds of the primitive. A simple solution is to simply mask the output of the linear layer per round. However, in order to reduce this need of dynamic randomness, we observe that it is sufficient to re-mask what an adversary can probe. The first probe's observed values would be re-masked by fresh randomness causing the second probe's values to necessarily be independent of the first one's. As a result, the security reverts back to the previous section on the first-order probing security. To provide an example on how this can be applied, consider an unpaired masked case for an AES. We want to add randomness after the linear layer in order to make the design multivariate secure. We consider what an adversary can observe when probing a masked S-box or a masked linear layer as shown in Figure 2. From the figure, we observe that a probing adversary is only capable of viewing one column of the state when placing one probe. As a result, when re-masking each column with the same randomness, we can ensure that every two S-boxes in the design (even in different rounds) have independent inputs. Namely, the values observed by the first probe are indeed re-masked meaning that the second probe can not observe related data. Another way of viewing this property is via the linear cryptanalytic framework by Beyne et al. [BDZ20] where the argument would be that there can not exist a non-zero correlation trail between any two S-boxes.

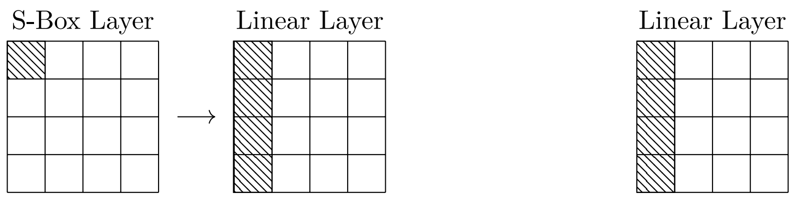


Figure 2: The output of the linear layer an adversary can observe by placing a probe in either one S-box (in the top left S-box) or in the linear layer (in the left ShiftRows and MixColumns operation) of a masked AES (where for the sake of an example, we don't consider the S-boxes are paired). Other probe positions are similar in that an adversary can at most observe one column of the state.

4 Case Studies

In this section, we apply the masking technique from Section 3 to three lightweight block ciphers MIDORI, SKINNY, and PRINCE. We confirm the security of the masked S-boxes with SILVER [KSM20] and have a theoretical analysis of multi-round probing security. We support our evaluations by practical experiments in Section 5.

4.1 Midori

As the first case study, we focus on MIDORI [BBI+15] which is a block cipher optimized for low-energy usage. The S-box is also used in other block ciphers including CRAFT [BLMR19] and MANTIS [BJK+16]. It has 128-bit key and 64-bit state that is split into 4-bit cells. An involutive binary quasi-MDS matrix together with a permutation of the 4-bit cells form the diffusion layer and it uses a 4-bit cubic S-box as the non-linear layer. MIDORI has a simple key schedule where each round either the left or right half of the master key is XORed to the state of the cipher.

Masking. MIDORI-64's S-box has three cubic coordinate functions and one quadratic one. Its lookup table is given by `cad3ebf789150246`. We can apply the technique expressed in Section 3.1 to find solutions for uniform sharings of each cubic coordinate functions. Since one coordinate function is quadratic, we need fewer coordinate functions. More precisely, 15 coordinate functions were enough to find a uniform sharing of the quadratic coordinate function. We found millions of uniform solutions for each coordinate function leading to a substantially large search space to find a joint-uniform solution. Therefore, we reduced the search space by removing some index configurations of each coordinate function. Looking at Table 1, we have four index configurations I_i that should be assigned to four input variables. We assign I_0 to the input variable that is common in all cubic monomials in the Algebraic Normal Form (ANF) of the target coordinate function. For example, the first coordinate function can be represented as $f(a, b, c, d) = b + ac + ad + abc + abd + bcd$. Since the input variable b exists in all cubic monomials, we assign the index configuration I_0 to b and assign other index configurations to the rest input variables randomly. If only one cubic monomial exists in the ANF of the function, we can freely choose any input variable in that cubic monomial to be assigned to I_0. We note that if no input variable can be identified, we search for all possible index configuration tables. In this way, we have to search using only one possible index configuration table per coordinate function which significantly reduces the search space size. Based on our observation, the probability of finding a joint-uniform sharing is also higher. However, we do not claim there is no joint-uniform sharing using other index configuration tables since the search space is too large to verify all possibilities.

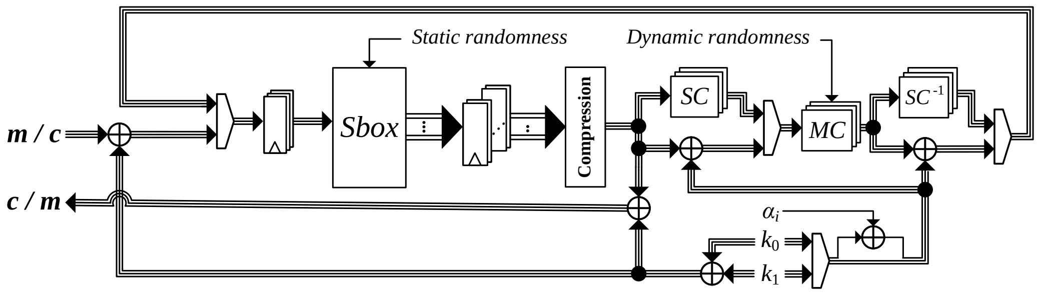


Figure 3: Design architecture of the round-based second-order secure Midori-64 encryption/decryption function.

To find a joint-uniform solution, we used the trick described in Section 3.1 to discard non-uniform combinations during the search. Namely, we started by finding a joint-uniform solution for two coordinate functions and then add the third and fourth one step-by-step checking the joint uniformity on the way. Our program, which ran on a machine with 24 CPU cores using 96 GB of RAM, found joint-uniform solutions in mere minutes.

To achieve first-order probing security, we paired two S-boxes as explained in Section 3.2. We added fresh masks to make the constructions second-order probing secure following the strategy discussed in Section 3.3. Notably, these fresh masks can be reused in all S-box pairs making it static randomness. Furthermore, these fresh masks stay unchanged during the execution of the cipher. One of the solutions is given in Appendix A.

Architecture. The design architecture of our round-based second-order secure Midori-64 implementation is depicted in Figure 3. The construction supports both encryption and decryption to make a fair comparison to the state of the art. It has two layers of registers; one right before the compression layer and one before the masked S-box. We used 96 fresh masks to make the S-box second-order secure. Since the S-boxes are paired, the randomness cost is doubled. In other words, a pair of S-boxes needs $96 \times 2 = 192$ fresh masks. These fresh masks can be reused in all other pairs in all rounds and we marked it in the figure as “static randomness”. They are given to the design at the start of the encryption and stay unchanged during the execution of the masked cipher. To ensure multivariate security we add 24-bits of fresh masks (dynamic randomness) after the diffusion layer of the cipher for the security reasons explained in Section 3.3. More specifically, we re-mask the columns using the random bytes $(r_1, r_2, r_3, r_1 + r_2 + r_3)$ (the fourth cell in a column is re-masked using the sum of the randomness of the other cells). We use the same randomness for each column in that round but refresh the randomness per round. In addition, we add 8-bits of fresh randomness (denoted r_0) on the second paired S-box. Meaning that the first S-box in a pair is not re-masked but the second one is, where the same randomness is used per pair in that round but is refreshed per round. The explanation for this choice of refreshing is given further on in the security analysis.

To compare our synthesis results to the state of the art, we refer to Table 3 where the randomness cost of sharing the plaintext and key is included . The design presented in [SM21] has high randomness complexity and requires four clock cycles per round. The implementation cost is reduced in [BDMS22] in terms of randomness and latency. However, each round is performed in three clock cycles. Namely, in both mentioned works, the authors dealt with quadratic functions and hence they had to decompose the S-box, resulting in higher latency in the number of clock cycles. However, our technique can be applied to cubic functions making us achieve lower latency (two clock cycles per round) while maintaining the same throughput and delay at the cost of a higher area overhead.

Table 3: Performance figures of different implementations.
(using Synopsis Design Compiler, and UMC 90 standard cell library, excluding RNGs)

Design	Security Order	No. of Shares	Fresh Masks/ Encryption [bit]	Area [kGE]	Delay [ns]	Latency [cycles]	Throughput [MB/s]
Midori-64							
[SM21]	2	3	8576	15.5	2.86	64	174.8
[BDMS22]	2	3	408	13.9	2.94	64	170.0
[BDMS22]	2	3	456	16.6	2.95	48	169.5
This work	2	3	1088	40.8	2.94	32	170.0
Skinny-64-64							
[SM21]	2	3	16640	10.6	1.22	128	204.9
[BDMS22]	2	3	296	12.4	1.33	128	188.0
[BDMS22]	2	3	320	12.3	1.33	96	188.0
This work	2	3	1424	25.5	2.24	64	111.6
Prince							
[BKN19][a]	2	3	21120	13.4	4.00	72	27.7
[BKN19][a]	2	5	12032	18.7	4.10	72	27.1
[BKN19][a,b]	2	3	41856	32.4	3.42	24	194.9
[BKN19][a,b]	2	8	34496	177.6	3.54	24	188.3
[SM21]	2	3	11136	19.4	3.11	84	214.3
[BDMS22]	2	3	454	19.5	3.08	72	216.4
[BDMS22]	2	3	584	20.3	3.41	48	195.5
This work	2	3	984	62.3	4.85	24	137.5

[a] using TSMC 90
[b] without S-box decomposition

The reason behind the higher area overhead is the fact that at least 27 coordinate functions are needed per cubic function to realize a second-order secure implementation using the minimum number of shares. Instead, the same level of security can be achieved with 9 coordinate functions for a quadratic function. The randomness cost of our design is significantly lower than the proposed design in [SM21], but higher than the construction in [BDMS22] as we have to introduce some fresh masks in the diffusion layer that should be updated each round. It is noteworthy to mention that since all designs in Table 3 are fully-pipelined, the throughput only rely on the delay (critical path) of the circuit and reducing the number of clock cycle does not lead to better throughput. Instead, as an advantage, the result would be ready to use in less number of clock cycles.

Algorithmic Security Analysis. As a first step, we evaluate the glitch-extended probing security of one round of the masked construction. To this end, we use the formal verification tool SILVER [KSM20]. The verification is split in two parts. For the first part, we removed all fresh masks from our paired S-box construction and checked the first-order probing security. For the second part, we verified the second-order probing security of one S-box which uses fresh randomness. However, because of using different fresh masks for the other S-box in the same pair, we can conclude that the entire construction of a paired S-box is second-order secure. It is important to follow this manner as the fresh masks that are used to achieve first-order security cannot be reused in our constructions. Let us assume that an intermediate variable is secured against a first-order attack by a single bit fresh mask that is reused in another part of the cipher. The adversary can observe the fresh mask by one probe and use the other probe to observe the sensitive variable whose security is dependent on that fresh mask. Hence, we first remove all fresh masks in the S-box construction - which will be reused - and verify the first-order security. The second-order verification has been done only for one S-box in a pair as the paired S-box is too large for the tool as a whole (as there are too many input variables considering the fresh masks for the tool). We should highlight that the two S-boxes in a pair receive completely different fresh masks so we can conclude that a paired construction is second-order secure.

We claim that placing two probes in two different pairs is also second-order secure while fresh masks are reused. Note that each pair is first-order secure with no fresh masks. One probe reveals at most all fresh masks and due to first-order security of a pair with no fresh masks, the second probe does not observe any sensitive variables. As a result, one round of the masked Midori is glitch-extended second-order probing secure.

For a second step, we show that the Midori is multivariate second-order probing secure. More specifically, that the design is secure in the case two probes are placed in different rounds of the cipher. This security is guaranteed by the use of dynamic randomness between the rounds as explained in Section 3.3. To show this security, we evaluate what an adversary can observe by placing a probe in a masked S-box pair or the linear layer. We overlap this by the dynamic randomness which is added to that layer. The result is shown in Figure 4.

Theorem 1. *The activity patterns caused by a glitch-extended probe (up to symmetry, the ones shown in Figure 4) are masked by the dynamic randomness of the masked Midori design from Section 4.1.*

Proof. We show that when one probe is placed in the round function of the masked Midori, the dynamic randomness refreshes what is observed. We split the reasoning in two parts, a probe is placed in the linear layer and a probe is placed in the masked S-box.

- We consider the case where a probe is placed in the linear layer. From the design of Midori's linear layer, this probe can view at most three cells of the state. Due to glitches, the adversary can view the compression layers of three S-boxes. Since this compression layer is fully masked by the paired S-box's inputs, the adversary can only view one of the paired S-boxes. As a result, the adversary views three cells of the state with the condition that there can be at most one cell active per row (an example is shown in Figure 4). It is clear that the dynamic randomness (r_1, r_2, r_3, r_4) with $r_4 = r_1 + r_2 + r_3$ refreshes this pattern.
- We consider the case where a probe is placed in the paired masked S-box which results in the adversary viewing the input of two S-boxes (an example is shown in Figure 4). The randomness r_0 refreshed one S-box in the pair. The leftover S-box cascades through the linear layer of Midori which, since the linear layer has branch number four, means at most three cells of the state after the linear layer are active (in one column). This reverts to the previous case where the randomness (r_1, r_2, r_3, r_4) re-masks this pattern.

□

From Theorem 1, we find that when two probes are placed in two separate rounds, their observed values are independently distributed. Since the shared functions are uniform, the observed values from each separate probe act as joint uniform randomness. As a result, the values returned by the two probes are joint uniform random. Thus, following the probing security model as introduced in Section 2.2, the shared Midori is second-order probing secure when the two probes are placed in two different rounds of the state.

4.2 SKINNY

For the second case study, we consider the Skinny tweakable block cipher [BJK+16]. In this work, we consider the Skinny-64-64 variant of the family which consists of a

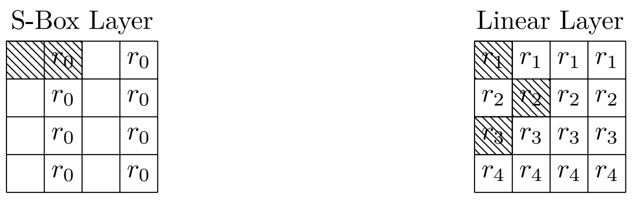


Figure 4: Examples of what an adversary can observe in the masked Midori (similarly in Skinny and Prince) by placing a probe in a masked S-box pair or in the linear layer. We include the dynamic randomness (with $r_4 = r_1 + r_2 + r_3$) which is added after the layers.

64-bit block size and a 64-bit tweakey. The state is divided in a 4x4 roster where each cell is a nibble. The S-box consists of a 4-bit cubic function and its linear layer consist of a ShiftRows and a MixColumns operation. More specifically, it uses a quasi-MDS MixColumns. The tweakey schedule from Skinny is linear and consists of a permutation and the application of a Linear Feedback Shift Register (LFSR).

Masking. The S-box of Skinny is given by the lookup table `c6901a2b385d4e7f`. It has two cubic coordinate functions and two quadratic ones. For the sharing, we started with the two quadratic coordinate functions. Since the ANF of both are simple, we could find a first-order probing secure (but non-uniform) sharing just by placing the shared monomials in 9 different coordinate functions for both of them. Then, we searched for a uniform sharing by adding linear correction terms to both and found a joint-uniform solution in just a few seconds. We should highlight that we did not use the pairing technique (from Section 3.2) for these two quadratic coordinate functions. The process for the two cubic coordinate functions is similar to the sharing of Midori. More precisely, we applied the technique expressed in Section 3 and found many solutions for each of them. As we had a joint-uniform solution for the quadratic coordinate functions, we first add the third coordinate function and check the joint uniformity. Once we found a solution for three coordinate functions, we added the solutions for the last coordinate function. This way, we found a joint-uniform solution in a couple of minutes for the Skinny S-box. By pairing two S-boxes (more precisely, only the cubic coordinate functions), we made the construction first-order glitch-extended probing secure. A total of 27 and 9 fresh masks are needed to secure the cubic and quadratic coordinate functions, respectively. As a result, each S-box requires $2 \times 27 + 2 \times 9 = 72$ fresh masks and a total of $2 \times 72 = 144$ fresh masks are needed for one pair. Again, these 144 fresh masks can be re-used in all other pairs in all rounds. The second-order probing secure masking of the Skinny S-box is given in detail in Appendix B. In summary, the above details a three-share second-order probing-secure realization of the SkinnyS-box with only one register layer using 144 static random bits.

Architecture. The general structure of our fully-pipelined round-based second-order Skinny-64 is illustrated in Figure 5, where each round is performed in only two clock cycles. Note that we had to place a layer of registers right before the S-box to maintain the second-order glitch-extended probing security of the entire design (this can be seen as the state register). As previously stated, 144 fresh masks are given to the design at the start of the encryption together with the shared key and the shared plaintext. The design also requires 32 fresh masks which should be updated each round to be introduced in the diffusion layer to achieve multivariate (multi-round) security. This addition of dynamic randomness follows the same design as used in our sharing of Midori. Namely, we refresh the second S-box of a pair (using 8-bits of randomness) and we refresh the columns after the linear layer (using 24 bits of randomness as the fourth cell's randomness can be the sum of the other three). Table 3 shows the corresponding performance figures, where the

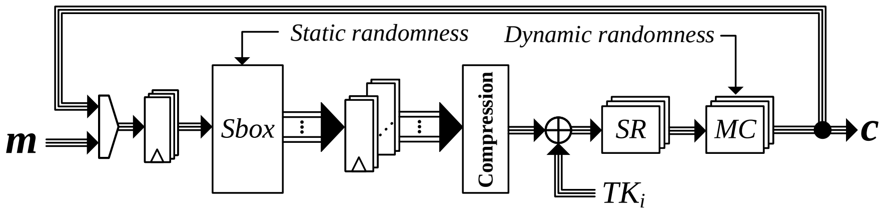


Figure 5: Design architecture of our round-based second-order SKINNY-64 encryption function.

required fresh masks for sharing the key and plaintext are included. We only implemented and reported the synthesis result for SKINNY-64-64 (64-bit state and 64-bit key) in order to make a fair comparison to the state of the art. Note that other variants with larger key sizes can be easily implemented due to the simplicity of the SKINNY key schedule. Our design outperforms the implementation presented in [SM21] as it needs significantly more fresh masks and double the latency. Compared to [BDMS22], we reduced the latency at the cost of more fresh masks and an area overhead due to us sharing cubic functions instead of quadratic ones. Since we removed register layers to achieve lower latency, the critical path (delay) of the circuit increased as well.

Algorithmic Security Analysis. We first start with the verification of our masked S-box using SILVER [KSM20]. We employed the same approach described in the security analysis of MIDORI. Namely, we first verified the first-order glitch-extended probing security of the paired S-boxes without fresh masks. This implies that two probes placed in two different paired S-boxes in the same round is secure. Second, we were able to confirm the second-order security of one S-box with fresh masks. As a result, the security of the case that the adversary place two probes in the same pair is also verified and thus a single round of our shared SKINNY is second-order probing secure. The verification took a day on a machine with 16 cores and 128 GB of RAM.

The multi-round probing security of the shared SKINNY follows from the use of dynamic randomness between the rounds. This is proven by going through what an adversary can observe using a single probe and tracking this activity pattern through the dynamic randomness separating the rounds. We then verify that the dynamic randomness covers the produced activity patterns. Since the activity patterns and the added randomness are the same as the one from the shared MIDORI, the same security arguments as in Section 4.1 can be used.

Theorem 2. *The activity patterns caused by a glitch-extended probe (up to symmetry, the ones shown in Figure 4) are masked by the dynamic randomness of the masked SKINNY design from Section 4.2.*

Proof. Since the patterns and the addition of randomness are exactly the same as in the case for the share MIDORI, the proof is the same as for Theorem 1. □

Similar to the MIDORI case, since two probes placed in two separate rounds return independent distributed data, the observed values from each probe follow a joint uniform random distribution. As a result, the shared SKINNY is second-order probing secure when the two probes are placed in two different rounds of the state.

4.3 PRINCE

For the third case study, we investigate a sharing of the PRINCE block cipher [BCG+12]. PRINCE is an AES-like cipher which consists of a 64-bit state divided in a 4x4 roster of nibbles and a 128-bit key. The S-box is a 4-bit cubic function and the linear layer consists of a ShiftRows and a MixColumns operation with a quasi-MDS matrix. The key schedule is simple where the first and last round key are the first 64-bits of the master key and the other round keys form the last 64-bits of the master key. The cipher consists of 12 nonlinear layers, where the first half applies the S-box and the second half applies the inverse S-box. Both the S-box and its inverse are affine equivalent.

Masking. Both the S-box (given by the lookup table `BF32AC916780E5D4`) and its inverse (given by the lookup table `B732FD89A6405EC1`) are used in the Prince block cipher in both the encryption and the decryption procedure. As a result, the application of our technique is not as straightforward as for Midori and Skinny. The algebraic degree of all coordinate functions of both the S-box and its inverse are three, making the search space to find a joint-uniform sharing large and its search non-trivial. Nevertheless, the Prince S-box and its inverse are affine equivalent. However, while this fact has proven useful in [MS16] and later in [SM21, BDMS22], it forces us to introduce several register stages to fulfill second-order glitch-extended probing security leading to a higher latency. Hence, the application of our strategy to only S-box (or its inverse) would lead to a construction where each round is performed in three clock cycles. While it is still better than the state-of-the-art, we intend to reduce it further down to two clock cycles per round. Therefore, we follow another design architecture presented in [SM20], where both the S-box and its inverse are implemented, as depicted in Figure 6. The compression layer is shared between the S-box and its inverse and is selected based on the cipher round. Since no register layer is placed before the multiplexer that selects either the S-box or its inverse, the probes that observes the output of the multiplexer expand backwards to all its inputs. As a result, both inputs of the multiplexer must fulfill the non-completeness property. In other words, the index shares that are used for a coordinate function of an S-box must be used in the corresponding coordinate functions of its inverse as well. This criterion forces us to use the same index configuration table per coordinate function. Following this approach, the corresponding coordinate functions of the S-box and its inverse plus the subsequent multiplexer in Figure 6 can be combined in a single module, to achieve a lower area overhead.

In order to reduce the search space's size, we first look at the cubic monomials of a particular coordinate function in both the S-box and its inverse and assign the index configuration I_0 to the common input variable of the cubic monomials. For example, the third coordinate function of the Prince S-box is $f(a,b,c,d) = a+ab+d+ad+bd+abd+bcd$ and the ANF representation of the corresponding coordinate function of the inverse S-box is $g(a,b,c,d) = a + ab + c + ac + bc + abc + bd + abd$. The input variable b is the only one that is used in all cubic monomials. Hence, we assign the index configuration I_0 to the input variable b and assign the rest randomly. This approach can be applied to three coordinate functions, however, there is no input variable that is common in all cubic monomials in the fourth coordinate function of the S-box and its inverse. Our program found over ten million joint-uniform solutions for these three coordinate functions for both the S-box and its inverse in only a couple of hours. For the fourth coordinate function, we considered all possible index configuration tables. We also considered a limited number of quadratic correction terms as well. Our program, running on a machine with 48 cores and 256 GB of RAM, could not find a joint-uniform solution after two weeks. Therefore, we had to use two different index configuration tables for this last function. Note that we can use multiplexer for two component functions that receive the same set of input. Since we had to use two different index configuration tables, we increased the number of coordinate functions to fulfill second-order non-completeness. For example, without violating second-order non-completeness, we can switch between two component functions $f_0(a_0, b_1, c_1, d_0)$ and $f_0''(a_0, b_1, c_1, d_0)$ using a multiplexer with not registers as their input

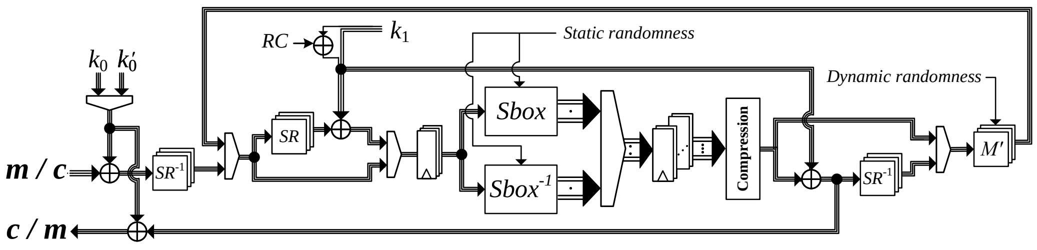


Figure 6: Design architecture of our round-based second-order PRINCE encryption/decryption function.

list are the same. However, that is not the case for $f_0''(a_1, b_1, c_1, d_0)$ since it violates the second-order non-completeness for input variable a. For this reason, we had to increase the number of component functions for the last coordinate functions of PRINCE S-box and its inverse. Namely, we increased it to 45 component functions instead of 27 where each 15 of them are compressed to generate one output share. One solution is given in detail in Appendix C.

We coupled two S-box/inverse S-box constructions to make it first-order glitch-extended probing secure and added $4 \times 27 = 108$ fresh masks to each construction (giving a total randomness cost of $2 \times 108 = 216$ bits for one pair) to achieve second-order glitch-extended probing security. These masks are static and thus can be reused in other pairs in all cipher rounds.

Architecture. The design architecture of our round-based second-order PRINCE is shown in Figure 6. We placed a state register right after the the first multiplexer (which also serves to ensure second-order probing security). The design requires 216 static masks and 32 dynamic ones. Recall that these dynamic masks should be updated each round. These dynamic masks re-mask the second S-box (or inverse S-box) in a pair (requiring 8 bits) and refresh the columns of the state (requiring 24 bits) similar to the constructions in our proposed MIDORI and SKINNY.

There are three other works proposing second-order masked hardware implementations of PRINCE. Table 3 shows a comparison between the performance of these designs. In [BKN19], the S-box is decomposed into quadratic bijections and some modules are reused in the implementation to reduce the area of the construction. The designs demand a high number of fresh masks and has a low throughput due to its serial architecture. The authors also presented two different constructions without a decomposition of the S-box at the cost of a significant randomness overhead. The number of fresh masks is reduced in [SM21], however, the latency increased. The low-latency design presented in [BDMS22] outperformed all mentioned designs in terms of fresh masks and latency with roughly the same area overhead. Still, in the low-latency design, each round is performed in four clock cycles. Whereas our design requires only two clock cycles while keeping the randomness cost low. Contrary to most proposed designs, we employed no S-box decomposition and implemented both the S-box and its inverse to achieve a lower latency leading to a higher area overhead. Since we removed register layers, the critical path of our construction is longer.

Algorithmic Security Analysis. We start with the security analysis of one round of the primitive. The verification has been done using SILVER [KSM20]. We followed the approach outlined in the security analysis of MIDORI and SKINNY and confirmed the first- and second-order glitch-extended probing security of our construction.

The multi-round probing security of the shared PRINCE follows from the use of dynamic randomness between the rounds. Since the linear layers in PRINCE have the same construction (same branch number and the use of a ShiftRows operation) as MIDORI and SKINNY, and since dynamic randomness is added in the same fashion as the other designs, its security follows similarly to the other cases.

Theorem 3. *The activity patterns caused by a glitch-extended probe (up to symmetry, the ones shown in Figure 4) are masked by the dynamic randomness of the masked PRINCE design from Section 4.3.*

Proof. Since the patterns and the addition of randomness are exactly the same as in the case for the share MIDORI, the proof is the same as for Theorem 1. □

Similar to the MIDORI and SKINNY cases, since two probes placed in two separate rounds return independent distributed data, the observed values from each probe follow a joint uniform random distribution. As a result, the shared PRINCE is second-order probing secure when the two probes are placed in two different rounds of the state.

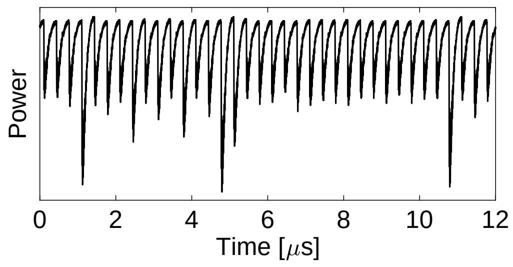
5 Experimental Analysis

As previously stated, we confirmed the security of all S-box constructions with SILVER [KSM20]. Since no tool can currently evaluate an entire cipher, we have conducted experimental analyses in addition to the theoretical analyses. For the first experimental analysis, we have taken our full cipher implementation of PRINCE. It requires 216 bits of static fresh masks and 32 bits of dynamic masks apart from the sharing of the plaintext and key. We implemented this design on a Xilinx Kintex-7 FPGA of the SAKURA-X evaluation board [SAK] which is supplied by a stable 6 MHz oscillator as the source for the clock. Each required fresh mask bit is provided by an LFSR with the feedback polynomial $x^{31} + x^{28} + 1$. As discussed in [DMW18], the LFSR can be efficiently implemented in Xilinx FPGAs using only three 6-to-1 Look-Up Tables (LUTs). The LFSRs dedicated to the dynamic fresh masks are updated at every clock cycle, but those that generate static fresh masks are activated for only one clock cycle for each encryption. Namely, they are only updated once at the start of encryption and remain constant for each given plaintext to be encrypted. The target FPGA receives a three-share masked input (plaintext) and a masked key and provides the output (ciphertext) also in a three-share masked form.

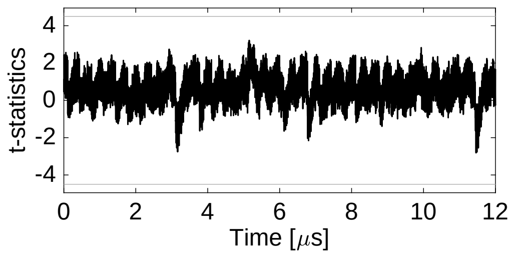
By means of a digital oscilloscope at a sampling rate of 500 MS/s, we collected power consumption traces while measuring the voltage drop over a $1\,\Omega$ shunt resistor placed on the VDD path of the target FPGA. We have conducted a fixed-versus-random t-test - also known as TVLA [CDG+13] - for each design using 100 million traces. In this test, the key is fixed to a certain value during the measurement and the masked design receives either a fixed or a random plaintext. Independent on this, all inputs are given in a three-share masked form with uniform sharing.

We have performed three different univariate analyses. The first one is an ordinary t-test, where the test is applied to each sample point one-by-one called a *first-order univariate test*. To apply a *second-order univariate test*, we need to pre-process the traces. Namely, for each group of fixed and random samples, individually, we made the traces mean-free followed by squaring each sample point. Afterwards, the same procedure for the first-order univariate test is followed. For the *third-order univariate test*, we performed the same pre-processing method as for the second-order univariate test while each mean-free sample was cubed instead of squared.

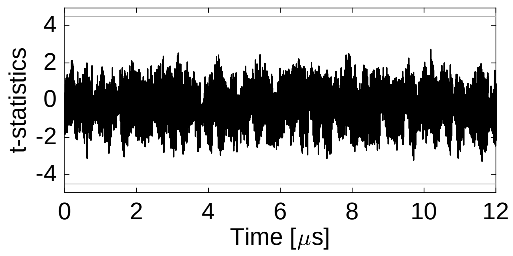
We have also conducted a *bivariate second-order t-test*. In this case, each combination of every two mean-free sample points should be multiplied, and an individual t-test should be performed for each combination leading to a significant number of individual tests.



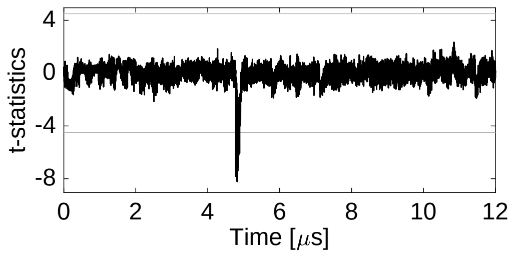
(a) A sample trace



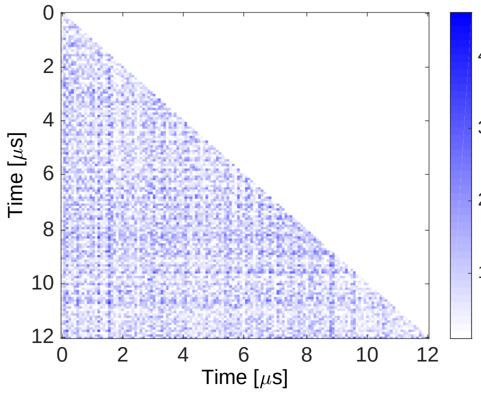
(b) 1st-order t-test



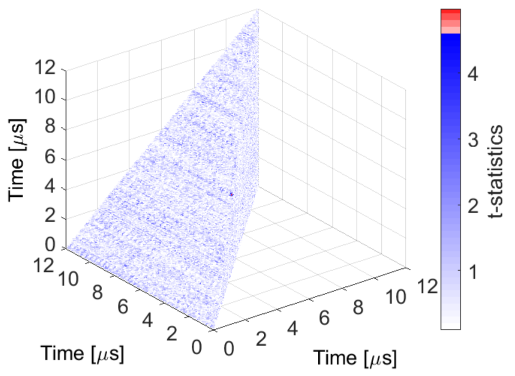
(c) 2nd-order t-test



(d) 3rd-order t-test



(e) 2nd-order bivariate t-test



(f) 3rd-order trivariate t-test

Figure 7: Experimental analysis of our masked Prince using 100 million traces.

Since this is not feasible in practice, we took four samples per clock cycle that are carefully selected to over start, end and middle of each clock cycle. Therefore, we were able to cover all clock cycles involved in the power traces. This downsampling trick has been also used in several other publications including [CRB$^+$16, SM21, BDMS22]. Note that, as discussed in [MOP07], power consumption traces are low-pass filtered by the Printed Circuit Board (PCB), the measurement facility, etc. As a result, the information on the leakage is roughly the same for several sample points in each clock cycle of the power traces. More discussion and information can be found in [MM13]. For the sake of completeness, we also performed a *trivariate second-order t-test*, where an individual t-test for each combination of every three possible sample points should be performed. For this analysis, we used the same downsampled traces as we used for the bivariate analysis, i.e., four samples per clock cycle.

The corresponding results are depicted in Figure 7, confirming the second-order security of our design. Our experimental analyses have not shown any first- or second-order leakage in both the univariate and bivariate tests. As presented in Figures 7d and 7f, the design exhibits third-order leakage in both univariate and multivariate scenarios. This confirms the ability of our setup to detect such higher-order leakages. The observed third-order leakage actually corresponds to the middle of the PRINCE encryption, i.e., the clock cycle when the multiplexers switch between the S-box and the inverse S-box modules (see Figure 6). Note that, the 128-bit key in PRINCE is split into two 64-bit parts and both are XORed to the plaintexts as the first operation. In the round that the leakage is detected, all bits of the key and the given plaintext are fully mixed. Hence, it is almost impossible to exploit this leakage and conduct a successful key-recovery since a large part of the key should be guessed (almost all 128 bits). Due to the reasons explained in Section 4.3, the area overhead of the S-box and S-box inverse is larger than other case studies followed by large multiplexers. As a result, this part of the cipher consumes more energy and would amplify the third-order leakage while switching from S-box to S-box inverse leading to detectable third-order leakage even in the univariate analysis.

We have also followed the same procedure for our MIDORI and SKINNY designs, where the corresponding results are given in Appendix D. In short, we do not detect any first-order or second-order leakage in the univariate or bivariate analyses. Contrary to PRINCE where the third-order leakage is detected in both univariate and bivariate analyses, we only have third-order leakage in the multivariate t-test of SKINNY.

For the sake of sanity check of our measurement setup, we evaluated the masked SKINNY design turning the PRNG off with an unshared input to emulate an unprotected implementation. We performed a t-test and detected first-order leakage as demonstrated in Figure 10. We also performed a first- and second-order t-test on SKINNY with the PRNG off but with a shared input. As stated in Section 4.2, the design is first-order secure with no fresh masks but not second-order secure. Our experimental analysis also confirms this claim, as depicted in Figure 11.

6 Discussions and Conclusions

In this work, we have introduced a methodology to create three-share second-order secure implementations of cubic functions with a low randomness cost. We then applied the methodology to create second-order secure hardware implementations of MIDORI, SKINNY, and PRINCE. To the best of our knowledge, our designs are the only ones that provide second-order security where each round of the cipher is performed in only two clock cycles without demanding a high number of fresh masks. More importantly, we confirmed the second-order glitch-extended probing security of our masked S-box constructions using the formal verification tool SILVER, we provided a theoretical analysis of the multi-round probing security of our designs, and we conducted experimental analyses to support our claims.

Apart from these achievements, we should highlight that our masked S-box's security have not been proven in the Non-Interferent (NI) [BBD+16] or Probe-Isolating Non-Interferent (PINI) [CS20] frameworks (since we use inputs from another S-box for re-masking purposes). Instead, extra randomness can be used to wall-off the masked primitives or the security of the larger whole can be carefully examined to save these extra costs.

The application of our methodology on the AES S-box is the most interesting case for future work given that it can be decomposed into cubic functions. However, this is challenging as the composition of the masked cubic functions may require a high number of fresh masks to ensure its security. Hence, this difficulty needs to be overcome to achieve better results compared to the state-of-the-art.

Another important line of future work is the investigation on the cost of randomness generation in hardware in terms of area, energy, power, and latency. Similar to other publications in the field, we did not consider the cost of the PRNGs which generate the fresh masks in the performance figures in Table 3. The generation of (SCA secure) fresh masks can be costly in hardware, particularly when they must be updated every clock cycles. Therefore, it would be beneficial to determine how expensive it is to generate them and what the best approach would be to lower area/energy/power/latency.

Acknowledgment. The work described in this paper has been supported in part by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972 and through the project 406956718 SuCCESS. Siemen Dhooghe is supported by a PhD Fellowship from the Research Foundation – Flanders (FWO).

References

[BBC$^+$19] Gilles Barthe, Sonia Belaïd, Gaëtan Cassiers, Pierre-Alain Fouque, Benjamin Grégoire, and François-Xavier Standaert. maskVerif: Automated Verification of Higher-Order Masking in Presence of Physical Defaults. In *ESORICS 2019*, volume 11735 of *Lecture Notes in Computer Science*, pages 300–318. Springer, 2019.

[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 *CCS 2016*, pages 116–129. ACM, 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 *ASIACRYPT 2015*, volume 9453 of *Lecture Notes in Computer Science*, 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, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. PRINCE - A Low-Latency Block Cipher for Pervasive Computing Applications - Extended Abstract. In *ASIACRYPT 2012*, volume 7658 of *Lecture Notes in Computer Science*, pages 208–225. Springer, 2012.

[BDMS22] Tim Beyne, Siemen Dhooghe, Amir Moradi, and Aein Rezaei Shahmirzadi. Cryptanalysis of efficient masked ciphers: Applications to low latency. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(1):679–721, 2022.

[BDZ20] Tim Beyne, Siemen Dhooghe, and Zhenda Zhang. Cryptanalysis of Masked Ciphers: A Not So Random Idea. In *ASIACRYPT 2020*, volume 12491 of *Lecture Notes in Computer Science*, pages 817–850. Springer, 2020.

[BGN$^+$14] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Higher-Order Threshold Implementations. In *ASIACRYPT 2014*, volume 8874 of *Lecture Notes in Computer Science*, pages 326–343. Springer, 2014.

[BGR18] Sonia Belaïd, Dahmun Goudarzi, and Matthieu Rivain. Tight Private Circuits: Achieving Probing Security with the Least Refreshing. In *ASIACRYPT 2018*, volume 11273 of *Lecture Notes in Computer Science*, pages 343–372. Springer, 2018.

[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 *CRYPTO*

2016, volume 9815 of *Lecture Notes in Computer Science*, pages 123–153. Springer, 2016.

[BKN19] Dusan Bozilov, Miroslav Knezevic, and Ventzislav Nikov. Optimized Threshold Implementations: Minimizing the Latency of Secure Cryptographic Accelerators. In *CARDIS 2019*, volume 11833 of *Lecture Notes in Computer Science*, pages 20–39. Springer, 2019.

[BLMR19] Christof Beierle, Gregor Leander, Amir Moradi, and Shahram Rasoolzadeh. CRAFT: Lightweight Tweakable Block Cipher with Efficient Protection Against DFA Attacks. *IACR Trans. Symmetric Cryptol.*, 2019(1):5–45, 2019.

[BNN$^+$15] Begül Bilgin, Svetla Nikova, Ventzislav Nikov, Vincent Rijmen, Natalia N. Tokareva, and Valeriya Vitkup. Threshold implementations of small S-boxes. *Cryptogr. Commun.*, 7(1):3–33, 2015.

[CBR$^+$15] Thomas De Cnudde, Begül Bilgin, Oscar Reparaz, Ventzislav Nikov, and Svetla Nikova. Higher-order threshold implementation of the AES s-box. In *CARDIS 2015*, volume 9514 of *Lecture Notes in Computer Science*, pages 259–272. Springer, 2015.

[CDG$^+$13] Jeremy Cooper, Elke DeMulder, Gilbert Goodwill, Joshua Jaffe, Gary Kenworthy, Pankaj Rohatgi, et al. Test vector leakage assessment (TVLA) methodology in practice. In *International Cryptographic Module Conference*, volume 20, 2013.

[CRB$^+$16] Thomas De Cnudde, Oscar Reparaz, Begül Bilgin, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Masking AES with d+1 Shares in Hardware. In *CHES 2016*, volume 9813 of *Lecture Notes in Computer Science*, pages 194–212. Springer, 2016.

[CS20] Gaëtan Cassiers and François-Xavier Standaert. Trivially and Efficiently Composing Masked Gadgets With Probe Isolating Non-Interference. *IEEE Trans. Information Forensics and Security*, 15:2542–2555, 2020.

[Dae17] Joan Daemen. Changing of the Guards: A Simple and Efficient Method for Achieving Uniformity in Threshold Sharing. In *CHES 2017*, volume 10529 of *Lecture Notes in Computer Science*, pages 137–153. Springer, 2017.

[DMW18] Lauren De Meyer, Amir Moradi, and Felix Wegener. Spin Me Right Round Rotational Symmetry for FPGA-Specific AES. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):596–626, 2018.

[DN22] Siemen Dhooghe and Svetla Nikova. Resilient uniformity: applying resiliency in masking. *Cryptogr. Commun.*, 14(1):41–58, 2022.

[DSM22] Siemen Dhooghe, Aein Rezaei Shahmirzadi, and Amir Moradi. Second-order low-randomness $d + 1$ hardware sharing of the aes. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, U.S.A., November 7-11, 2022*. ACM, 2022.

[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.

[GM18] Hannes Groß and Stefan Mangard. A unified masking approach. *J. Cryptogr. Eng.*, 8(2):109–124, 2018.

[GMK16] Hannes Groß, Stefan Mangard, and Thomas Korak. Domain-Oriented Masking: Compact Masked Hardware Implementations with Arbitrary Protection Order. In *Theory of Implementation Security - TIS@CCS 2016*, page 3. ACM, 2016.

[GPPR11] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The LED block cipher. volume 6917 of *Lecture Notes in Computer Science*, pages 326–341. Springer, 2011.

[ISW03] Yuval Ishai, Amit Sahai, and David A. Wagner. Private Circuits: Securing Hardware against Probing Attacks. In *CRYPTO 2003*, 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 *CRYPTO 1999*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[KM22a] David Knichel and Amir Moradi. Composable gadgets with reused fresh masks first-order probing-secure hardware circuits with only 6 fresh masks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(3):114–140, 2022.

[KM22b] David Knichel and Amir Moradi. Low-latency hardware private circuits. 2022.

[KSM20] David Knichel, Pascal Sasdrich, and Amir Moradi. SILVER - Statistical Independence and Leakage Verification. In *ASIACRYPT 2020*, volume 12491 of *Lecture Notes in Computer Science*, pages 787–816. Springer, 2020.

[MBR19] Lauren De Meyer, Begül Bilgin, and Oscar Reparaz. Consolidating Security Notions in Hardware Masking. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(3):119–147, 2019.

[MM13] Amir Moradi and Oliver Mischke. On the Simplicity of Converting Leakages from Multivariate to Univariate - (Case Study of a Glitch-Resistant Masking Scheme). In *CHES 2013*, volume 8086 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2013.

[MME10] Amir Moradi, Oliver Mischke, and Thomas Eisenbarth. Correlation-Enhanced Power Analysis Collision Attack. In *CHES 2010*, volume 6225 of *Lecture Notes in Computer Science*, pages 125–139. Springer, 2010.

[MOP07] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks - revealing the secrets of smart cards.* Springer, 2007.

[MPL$^+$11] Amir Moradi, Axel Poschmann, San Ling, Christof Paar, and Huaxiong Wang. Pushing the Limits: A Very Compact and a Threshold Implementation of AES. In *EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 69–88. Springer, 2011.

[MPO05] Stefan Mangard, Norbert Pramstaller, and Elisabeth Oswald. Successfully Attacking Masked AES Hardware Implementations. In *CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 157–171. Springer, 2005.

[MS16] Amir Moradi and Tobias Schneider. Side-Channel Analysis Protection and Low-Latency in Action - - Case Study of PRINCE and Midori -. In *ASIACRYPT 2016*, volume 10031 of *Lecture Notes in Computer Science*, pages 517–547, 2016.

[NRR06] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold Implementations Against Side-Channel Attacks and Glitches. In *ICICS 2006*, volume 4307 of *Lecture Notes in Computer Science*, pages 529–545. Springer, 2006.

[NRS11] Svetla Nikova, Vincent Rijmen, and Martin Schläffer. Secure Hardware Implementation of Nonlinear Functions in the Presence of Glitches. *J. Cryptology*, 24(2):292–321, 2011.

[RBN$^+$15] Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating Masking Schemes. In *CRYPTO 2015*, volume 9215 of *Lecture Notes in Computer Science*, pages 764–783. Springer, 2015.

[Rep15] Oscar Reparaz. A note on the security of Higher-Order Threshold Implementations. *IACR Cryptol. ePrint Arch.*, 2015:1, 2015.

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

[SBM21] Aein Rezaei Shahmirzadi, Dusan Bozilov, and Amir Moradi. New first-order secure AES performance records. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(2):304–327, 2021.

[SM20] Aein Rezaei Shahmirzadi and Amir Moradi. Re-Consolidating First-Order Masking Schemes - Nullifying Fresh Randomness. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(1):305–342, 2020.

[SM21] Aein Rezaei Shahmirzadi and Amir Moradi. Second-order SCA security with almost no fresh randomness. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(3):708–755, 2021.

[Tri03] Elena Trichina. Combinational Logic Design for AES SubByte Transformation on Masked Data. *IACR Cryptol. ePrint Arch.*, 2003:236, 2003.

[ZSS$^+$21] Sara Zarei, Aein Rezaei Shahmirzadi, Hadi Soleimany, Raziyeh Salarifard, and Amir Moradi. Low-latency keccak at any arbitrary order. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(4):388–411, 2021.

A 3-share Midori S-Box with 96-bit Fresh Masks

$F(a,\ b,\ c,\ d) = (x,\ y,\ z,\ t)$ with lookup table `cad3ebf789150246`
$x = f(a,b,c,d) = b + ac + ad + abc + abd + bcd$
$y = g(a,b,c,d) = a + c + ac + ad + cd$
$z = h(a,b,c,d) = 1 + a + d + ad + abc + abd + bcd$
$t = u(a,b,c,d) = 1 + ab + bd + cd + abd + bcd$

We denote the uniform sharing of the function in black.
$(k,\ l,\ m,\ n)$ are the input variables of the paired S-box, which are denoted in green.
Fresh masks are denoted in red.

$$
\begin{array}{llll}
f_0(a_0,b_0,c_0,d_0) & = b_0a_0c_0 + b_0a_0d_0 + b_0c_0d_0 + a_0d_0 + a_0c_0 + k_0 + r_0 + r_1 & \to x'_0 & \\
f_1(a_0,b_0,c_1,d_1) & = b_0a_0c_1 + b_0a_0d_1 + b_0c_1d_1 + a_0d_1 + d_1 + a_0 + k_0 + l_0 + r_1 + r_2 & \to x'_1 & \\
f_2(a_0,b_0,c_2,d_2) & = b_0a_0c_2 + b_0a_0d_2 + b_0c_2d_2 + a_0d_2 + c_2 + l_0 + m_0 + r_2 + r_3 & \to x'_2 & \\
f_3(a_1,b_0,c_1,d_0) & = b_0a_1c_1 + b_0a_1d_0 + b_0c_1d_0 + c_1 + a_1 + d_0 + m_0 + n_0 + r_3 + r_4 & \to x'_3 & \\
f_4(a_1,b_0,c_2,d_1) & = b_0a_1c_2 + b_0a_1d_1 + b_0c_2d_1 + +n_0 + k_1 + r_4 + r_5 & \to x'_4 & \bigoplus_{i=0}^{8} x'_i = x_0 \\
f_5(a_1,b_0,c_0,d_2) & = b_0a_1c_0 + b_0a_1d_2 + b_0c_0d_2 + a_1c_0 + c_0 + k_1 + l_1 + r_5 + r_6 & \to x'_5 & \\
f_6(a_2,b_0,c_0,d_1) & = b_0a_2c_0 + b_0a_2d_1 + b_0c_0d_1 + a_2c_0 + l_1 + m_1 + r_6 + r_7 & \to x'_6 & \\
f_7(a_2,b_0,c_1,d_2) & = b_0a_2c_1 + b_0a_2d_2 + b_0c_1d_2 + a_2 + m_1 + n_1 + r_7 + r_8 & \to x'_7 & \\
f_8(a_2,b_0,c_2,d_0) & = b_0a_2c_2 + b_0a_2d_0 + b_0c_2d_0 + b_0 + n_1 + r_8 + r_9 & \to x'_8 & \\
\hline
f_9(a_0,b_1,c_2,d_0) & = b_1a_0c_2 + b_1a_0d_0 + b_1c_2d_0 + b_1 + k_0 + r_9 + r_{10} & \to x'_9 & \\
f_{10}(a_0,b_1,c_0,d_1) & = b_1a_0c_0 + b_1a_0d_1 + b_1c_0d_1 + +d_1 + c_0 + k_0 + l_0 + r_{10} + r_{11} & \to x'_{10} & \\
f_{11}(a_0,b_1,c_1,d_2) & = b_1a_0c_1 + b_1a_0d_2 + b_1c_1d_2 + c_1 + a_0 + a_0c_1 + l_0 + m_0 + r_{11} + r_{12} & \to x'_{11} & \\
f_{12}(a_1,b_1,c_2,d_2) & = b_1a_1c_2 + b_1a_1d_2 + b_1c_2d_2 + a_1d_2 + m_0 + n_0 + r_{12} + r_{13} & \to x'_{12} & \\
f_{13}(a_1,b_1,c_0,d_0) & = b_1a_1c_0 + b_1a_1d_0 + b_1c_0d_0 + a_1d_0 + n_0 + k_1 + r_{13} + r_{14} & \to x'_{13} & \bigoplus_{i=9}^{17} x'_i = x_1 \\
f_{14}(a_1,b_1,c_1,d_1) & = b_1a_1c_1 + b_1a_1d_1 + b_1c_1d_1 + a_1d_1 + a_1c_1 + k_1 + l_1 + r_{14} + r_{15} & \to x'_{14} & \\
f_{15}(a_2,b_1,c_0,d_2) & = b_1a_2c_0 + b_1a_2d_2 + b_1c_0d_2 + l_1 + m_1 + r_{15} + r_{16} & \to x'_{15} & \\
f_{16}(a_2,b_1,c_1,d_0) & = b_1a_2c_1 + b_1a_2d_0 + b_1c_1d_0 + a_2c_1 + m_1 + n_1 + r_{16} + r_{17} & \to x'_{16} & \\
f_{17}(a_2,b_1,c_2,d_1) & = b_1a_2c_2 + b_1a_2d_1 + b_1c_2d_1 + c_2 + n_1 + r_{17} + r_{18} & \to x'_{17} & \\
\hline
f_{18}(a_0,b_2,c_0,d_2) & = b_2a_0c_0 + b_2a_0d_2 + b_2c_0d_2 + b_2 + k_0 + r_{18} + r_{19} & \to x'_{18} & \\
f_{19}(a_0,b_2,c_1,d_0) & = b_2a_0c_1 + b_2a_0d_0 + b_2c_1d_0 + d_0 + k_0 + l_0 + r_{19} + r_{20} & \to x'_{19} & \\
f_{20}(a_0,b_2,c_2,d_1) & = b_2a_0c_2 + b_2a_0d_1 + b_2c_2d_1 + a_0c_2 + l_0 + m_0 + r_{20} + r_{21} & \to x'_{20} & \\
f_{21}(a_1,b_2,c_0,d_1) & = b_2a_1c_0 + b_2a_1d_1 + b_2c_0d_1 + m_0 + n_0 + r_{21} + r_{22} & \to x'_{21} & \\
f_{22}(a_1,b_2,c_2,d_0) & = b_2a_1c_2 + b_2a_1d_0 + b_2c_2d_0 + a_1c_2 + n_0 + k_1 + r_{22} + r_{23} & \to x'_{22} & \bigoplus_{i=18}^{26} x'_i = x_2 \\
f_{23}(a_1,b_2,c_1,d_2) & = b_2a_1c_1 + b_2a_1d_2 + b_2c_1d_2 + a_1 + k_1 + l_1 + r_{23} + r_{24} & \to x'_{23} & \\
f_{24}(a_2,b_2,c_1,d_1) & = b_2a_2c_1 + b_2a_2d_1 + b_2c_1d_1 + a_2d_1 + a_2 + l_1 + m_1 + r_{24} + r_{25} & \to x'_{24} & \\
f_{25}(a_2,b_2,c_0,d_0) & = b_2a_2c_0 + b_2a_2d_0 + b_2c_0d_0 + a_2d_0 + m_1 + n_1 + r_{25} + r_{26} & \to x'_{25} & \\
f_{26}(a_2,b_2,c_2,d_2) & = b_2a_2c_2 + b_2a_2d_2 + b_2c_2d_2 + a_2d_2 + a_2c_2 + n_1 + r_{26} + r_0 & \to x'_{26} & \\
\end{array}
$$

$$
\begin{array}{llll}
g_0(a_0,c_0,d_0) & = a_0c_0 + a_0d_0 + c_0d_0 + k_0 + l_0 + r_{27} + r_{28} & \to y'_0 & \\
g_1(a_0,c_1,d_1) & = a_0c_1 + a_0d_1 + a_0 + l_0 + m_0 + r_{28} + r_{29} & \to y'_1 & \\
g_2(a_0,c_2,d_2) & = a_0c_2 + a_0d_2 + m_0 + k_1 + r_{29} + r_{30} & \to y'_2 & \bigoplus_{i=0}^{4} y'_i = y_0 \\
g_3(c_0,d_1) & = c_0d_1 + c_0 + n_0 + k_0 + r_{30} + r_{31} & \to y'_3 & \\
g_4(b_2,c_0,d_2) & = c_0d_2 + b_2 + n_0 + k_1 + r_{31} + r_{32} & \to y'_4 & \\
\hline
g_5(a_1,c_1,d_1) & = a_1c_1 + a_1d_1 + c_1d_1 + k_0 + l_0 + r_{32} + r_{33} & \to y'_0 & \\
g_6(a_1,c_0,d_0) & = a_1c_0 + a_1d_0 + c_0 + l_0 + m_0 + r_{33} + r_{34} & \to y'_1 & \\
g_7(a_1,c_2,d_2) & = a_1c_2 + a_1d_2 + a_1 + m_0 + k_1 + r_{34} + r_{35} & \to y'_2 & \bigoplus_{i=5}^{9} y'_i = y_1 \\
g_8(c_1,d_0) & = c_1d_0 + d_0 + n_0 + k_0 + r_{35} + r_{36} & \to y'_3 & \\
g_9(c_1,d_2) & = c_1d_2 + c_1 + n_0 + k_1 + r_{36} + r_{37} & \to y'_4 & \\
\hline
g_{10}(a_2,c_2,d_2) & = a_2c_2 + a_2d_2 + c_2d_2 + k_0 + l_0 + r_{37} + r_{38} & \to y'_0 & \\
g_{11}(a_2,c_0,d_0) & = a_2c_0 + a_2d_0 + c_0 + a_2 + l_0 + m_0 + r_{38} + r_{39} & \to y'_1 & \\
g_{12}(a_2,c_1,d_1) & = a_2c_1 + a_2d_1 + m_0 + k_1 + r_{39} + r_{40} & \to y'_2 & \bigoplus_{i=10}^{14} y'_i = y_2 \\
g_{13}(c_2,d_0) & = c_2d_0 + d_0 + c_2 + n_0 + k_0 + r_{40} + r_{41} & \to y'_3 & \\
g_{14}(b_2,c_2,d_1) & = c_2d_1 + b_2 + n_0 + k_1 + r_{41} + r_{27} & \to y'_4 & \\
\end{array}
$$

$$
\begin{array}{llll}
h_0(a_0,b_0,c_0,d_0) & = b_0a_0c_0 + b_0a_0d_0 + b_0c_0d_0 + a_0d_0 + k_0 + r_{42} + r_{43} & \rightarrow z'_0 & \\
h_1(a_0,b_0,c_1,d_1) & = b_0a_0c_1 + b_0a_0d_1 + b_0c_1d_1 + a_0d_1 + d_1 + k_0 + l_0 + r_{43} + r_{44} & \rightarrow z'_1 & \\
h_2(a_0,b_0,c_2,d_2) & = b_0a_0c_2 + b_0a_0d_2 + b_0c_2d_2 + a_0d_2 + a_0 + l_0 + m_0 + r_{44} + r_{45} & \rightarrow z'_2 & \\
h_3(a_1,b_0,c_1,d_0) & = b_0a_1c_1 + b_0a_1d_0 + b_0c_1d_0 + d_0 + m_0 + n_0 + r_{45} + r_{46} & \rightarrow z'_3 & \\
h_4(a_1,b_0,c_2,d_1) & = b_0a_1c_2 + b_0a_1d_1 + b_0c_2d_1 + n_0 + k_1 + r_{46} + r_{47} & \rightarrow z'_4 & \bigoplus_{i=0}^{8} z'_i = z_0 \\
h_5(a_1,b_0,c_0,d_2) & = b_0a_1c_0 + b_0a_1d_2 + b_0c_0d_2 + c_0 + k_1 + l_1 + r_{47} + r_{48} & \rightarrow z'_5 & \\
h_6(a_2,b_0,c_0,d_1) & = b_0a_2c_0 + b_0a_2d_1 + b_0c_0d_1 + l_1 + m_1 + r_{48} + r_{49} & \rightarrow z'_6 & \\
h_7(a_2,b_0,c_1,d_2) & = b_0a_2c_1 + b_0a_2d_2 + b_0c_1d_2 + m_1 + n_1 + r_{49} + r_{50} & \rightarrow z'_7 & \\
h_8(a_2,b_0,c_2,d_0) & = b_0a_2c_2 + b_0a_2d_0 + b_0c_2d_0 + a_2 + n_1 + r_{50} + r_{51} & \rightarrow z'_8 & \\
\hline
h_9(a_0,b_1,c_2,d_0) & = b_1a_0c_2 + b_1a_0d_0 + b_1c_2d_0 + d_0 + 1 + k_0 + r_{51} + r_{52} & \rightarrow z'_9 & \\
h_{10}(a_0,b_1,c_0,d_1) & = b_1a_0c_0 + b_1a_0d_1 + b_1c_0d_1 + k_0 + l_0 + r_{52} + r_{53} & \rightarrow z'_{10} & \\
h_{11}(a_0,b_1,c_1,d_2) & = b_1a_0c_1 + b_1a_0d_2 + b_1c_1d_2 + l_0 + m_0 + r_{53} + r_{54} & \rightarrow z'_{11} & \\
h_{12}(a_1,b_1,c_2,d_2) & = b_1a_1c_2 + b_1a_1d_2 + b_1c_2d_2 + a_1d_2 + m_0 + n_0 + r_{54} + r_{55} & \rightarrow z'_{12} & \\
h_{13}(a_1,b_1,c_0,d_0) & = b_1a_1c_0 + b_1a_1d_0 + b_1c_0d_0 + a_1d_0 + n_0 + k_1 + r_{55} + r_{56} & \rightarrow z'_{13} & \bigoplus_{i=9}^{17} z'_i = z_1 \\
h_{14}(a_1,b_1,c_1,d_1) & = b_1a_1c_1 + b_1a_1d_1 + b_1c_1d_1 + a_1d_1 + a_1 + k_1 + l_1 + r_{56} + r_{57} & \rightarrow z'_{14} & \\
h_{15}(a_2,b_1,c_0,d_2) & = b_1a_2c_0 + b_1a_2d_2 + b_1c_0d_2 + l_1 + m_1 + r_{57} + r_{58} & \rightarrow z'_{15} & \\
h_{16}(a_2,b_1,c_1,d_0) & = b_1a_2c_1 + b_1a_2d_0 + b_1c_1d_0 + m_1 + n_1 + r_{58} + r_{59} & \rightarrow z'_{16} & \\
h_{17}(a_2,b_1,c_2,d_1) & = b_1a_2c_2 + b_1a_2d_1 + b_1c_2d_1 + a_2 + n_1 + r_{59} + r_{60} & \rightarrow z'_{17} & \\
\hline
h_{18}(a_0,b_2,c_0,d_2) & = b_2a_0c_0 + b_2a_0d_2 + b_2c_0d_2 + c_0 + k_0 + r_{60} + r_{61} & \rightarrow z'_{18} & \\
h_{19}(a_0,b_2,c_1,d_0) & = b_2a_0c_1 + b_2a_0d_0 + b_2c_1d_0 + d_0 + k_0 + l_0 + r_{61} + r_{62} & \rightarrow z'_{19} & \\
h_{20}(a_0,b_2,c_2,d_1) & = b_2a_0c_2 + b_2a_0d_1 + b_2c_2d_1 + l_0 + m_0 + r_{62} + r_{63} & \rightarrow z'_{20} & \\
h_{21}(a_1,b_2,c_0,d_1) & = b_2a_1c_0 + b_2a_1d_1 + b_2c_0d_1 + m_0 + n_0 + r_{63} + r_{64} & \rightarrow z'_{21} & \\
h_{22}(a_1,b_2,c_2,d_0) & = b_2a_1c_2 + b_2a_1d_0 + b_2c_2d_0 + n_0 + k_1 + r_{64} + r_{65} & \rightarrow z'_{22} & \bigoplus_{i=18}^{26} z'_i = z_2 \\
h_{23}(a_1,b_2,c_1,d_2) & = b_2a_1c_1 + b_2a_1d_2 + b_2c_1d_2 + k_1 + l_1 + r_{65} + r_{66} & \rightarrow z'_{23} & \\
h_{24}(a_2,b_2,c_1,d_1) & = b_2a_2c_1 + b_2a_2d_1 + b_2c_1d_1 + a_2d_1 + l_1 + m_1 + r_{66} + r_{67} & \rightarrow z'_{24} & \\
h_{25}(a_2,b_2,c_0,d_0) & = b_2a_2c_0 + b_2a_2d_0 + b_2c_0d_0 + a_2d_0 + m_1 + n_1 + r_{67} + r_{68} & \rightarrow z'_{25} & \\
h_{26}(a_2,b_2,c_2,d_2) & = b_2a_2c_2 + b_2a_2d_2 + b_2c_2d_2 + a_2d_2 + a_2 + d_2 + n_1 + r_{68} + r_{42} & \rightarrow z'_{26} &
\end{array}
$$

$$
\begin{array}{llll}
u_0(a_0,b_0,c_0,d_0) & = b_0a_0d_0 + b_0c_0d_0 + c_0d_0 + c_0 + k_0 + r_{69} + r_{70} & \rightarrow t'_0 & \\
u_1(a_0,b_0,c_1,d_1) & = b_0a_0d_1 + b_0c_1d_1 + c_1 + b_0a_0 + b_0d_1 + c_1d_1 + k_0 + l_0 + r_{70} + r_{71} & \rightarrow t'_1 & \\
u_2(a_0,b_0,c_2,d_2) & = b_0a_0d_2 + b_0c_2d_2 + b_0d_2 + c_2d_2 + l_0 + m_0 + r_{71} + r_{72} & \rightarrow t'_2 & \\
u_3(a_1,b_0,c_1,d_0) & = b_0a_1d_0 + b_0c_1d_0 + b_0a_1 + c_1d_0 + b_0d_0 + m_0 + n_0 + r_{72} + r_{73} & \rightarrow t'_3 & \\
u_4(a_1,b_0,c_2,d_1) & = b_0a_1d_1 + b_0c_2d_1 + n_0 + k_1 + r_{73} + r_{74} & \rightarrow t'_4 & \bigoplus_{i=0}^{8} t'_i = t_0 \\
u_5(a_1,b_0,c_0,d_2) & = b_0a_1d_2 + b_0c_0d_2 + c_0 + k_1 + l_1 + r_{74} + r_{75} & \rightarrow t'_5 & \\
u_6(a_2,b_0,c_0,d_1) & = b_0a_2d_1 + b_0c_0d_1 + c_0d_1 + l_1 + m_1 + r_{75} + r_{76} & \rightarrow t'_6 & \\
u_7(a_2,b_0,c_1,d_2) & = b_0a_2d_2 + b_0c_1d_2 + m_1 + n_1 + r_{76} + r_{77} & \rightarrow t'_7 & \\
u_8(a_2,b_0,c_2,d_0) & = b_0a_2d_0 + b_0c_2d_0 + b_0a_2 + c_2d_0 + c_2d_0 + n_1 + r_{77} + r_{78} & \rightarrow t'_8 & \\
\hline
tu_9(a_0,b_1,c_2,d_0) & = b_1a_0d_0 + b_1c_2d_0 + b_1a_0 + b_1d_0 + a_0 + c_2d_0 + k_0 + r_{78} + r_{79} & \rightarrow t'_9 & \\
u_{10}(a_0,b_1,c_0,d_1) & = b_1a_0d_1 + b_1c_0d_1 + c_0d_1 + b_1d_1 + a_0c_0 + c_0d_1 + k_0 + l_0 + r_{79} + r_{80} & \rightarrow t'_{10} & \\
u_{11}(a_0,b_1,c_1,d_2) & = b_1a_0d_2 + b_1c_1d_2 + b_1d_2 + a_0c_1 + l_0 + m_0 + r_{80} + r_{81} & \rightarrow t'_{11} & \\
u_{12}(a_1,b_1,c_2,d_2) & = b_1a_1d_2 + b_1c_2d_2 + b_1a_1 + a_1 + a_1c_2 + c_2d_2 + m_0 + n_0 + r_{81} + r_{82} & \rightarrow t'_{12} & \\
u_{13}(a_1,b_1,c_0,d_0) & = b_1a_1d_0 + b_1c_0d_0 + n_0 + k_1 + r_{82} + r_{83} & \rightarrow t'_{13} & \bigoplus_{i=9}^{17} t'_i = t_1 \\
u_{14}(a_1,b_1,c_1,d_1) & = b_1a_1d_1 + b_1c_1d_1 + c_1d_1 + c_1d_1 + k_1 + l_1 + r_{83} + r_{84} & \rightarrow t'_{14} & \\
u_{15}(a_2,b_1,c_0,d_2) & = b_1a_2d_2 + b_1c_0d_2 + l_1 + m_1 + r_{84} + r_{85} & \rightarrow t'_{15} & \\
u_{16}(a_2,b_1,c_1,d_0) & = b_1a_2d_0 + b_1c_1d_0 + m_1 + n_1 + r_{85} + r_{86} & \rightarrow t'_{16} & \\
u_{17}(a_2,b_1,c_2,d_1) & = b_1a_2d_1 + b_1c_2d_1 + b_1a_2 + c_2d_1 + a_2c_2 + n_1 + r_{86} + r_{87} & \rightarrow t'_{17} & \\
\hline
u_{18}(a_0,b_2,c_0,d_2) & = b_2a_0d_2 + b_2c_0d_2 + b_2a_0 + c_0d_2 + b_2d_2 + a_0 + a_0c_0 + k_0 + r_{87} + r_{88} & \rightarrow t'_{18} & \\
u_{19}(a_0,b_2,c_1,d_0) & = b_2a_0d_0 + b_2c_1d_0 + b_2d_0 + a_0c_1 + k_0 + l_0 + r_{88} + r_{89} & \rightarrow t'_{19} & \\
u_{20}(a_0,b_2,c_2,d_1) & = b_2a_0d_1 + b_2c_2d_1 + b_2d_1 + l_0 + m_0 + r_{89} + r_{90} & \rightarrow t'_{20} & \\
u_{21}(a_1,b_2,c_0,d_1) & = b_2a_1d_1 + b_2c_0d_1 + b_2a_1 + m_0 + n_0 + r_{90} + r_{91} & \rightarrow t'_{21} & \\
u_{22}(a_1,b_2,c_2,d_0) & = b_2a_1d_0 + b_2c_2d_0 + +a_1c_2 + n_0 + k_1 + r_{91} + r_{92} & \rightarrow t'_{22} & \bigoplus_{i=18}^{26} t'_i = t_2 \\
u_{23}(a_1,b_2,c_1,d_2) & = b_2a_1d_2 + b_2c_1d_2 + c_1d_2 + a_1 + c_1 + k_1 + l_1 + r_{92} + r_{93} & \rightarrow t'_{23} & \\
u_{24}(a_2,b_2,c_1,d_1) & = b_2a_2d_1 + b_2c_1d_1 + l_1 + m_1 + r_{93} + r_{94} & \rightarrow t'_{24} & \\
u_{25}(a_2,b_2,c_0,d_0) & = b_2a_2d_0 + b_2c_0d_0 + 1'b1 + m_1 + n_1 + r_{94} + r_{95} & \rightarrow t'_{25} & \\
u_{26}(a_2,b_2,c_2,d_2) & = b_2a_2d_2 + b_2c_2d_2 + b_2a_2 + c_2d_2 + a_2c_2 + n_1 + r_{95} + r_{69} & \rightarrow t'_{26} &
\end{array}
$$

B 3-share Skinny S-Box with 72-bit Fresh Masks

$F(a,\ b,\ c,\ d) = (x,\ y,\ z,\ t)$ with lookup table `c6901a2b385d4e7f`

$x = f(a,b,c,d) = b + ab + c + ac + abc + d + ad + bd + bcd$

$y = g(a,b,c,d) = a + ab + bc + d + bd + cd + bcd$

$z = h(a,b,c,d) = 1 + b + c + bc + d$

$t = u(a,b,c,d) = 1 + a + c + d + cd$

We denote the uniform sharing of the function in black.

$(k,\ l,\ m,\ n)$ are the input variables of the paired S-box, which are denoted in green.

Fresh masks are denoted in red.

$$\begin{aligned}
f_0(a_0,b_0,c_0,d_0) &= b_0a_0c_0 + b_0c_0d_0 + b_0a_0 + c_0a_0 + a_0d_0 + c_0 + d_0 + k_0 + r_0 + r_1 && \rightarrow x'_0\\
f_1(a_0,b_0,c_1,d_1) &= b_0a_0c_1 + b_0c_1d_1 + b_0d_1 + a_0 + k_0 + l_0 + r_1 + r_2 && \rightarrow x'_1\\
f_2(a_0,b_0,c_2,d_2) &= b_0a_0c_2 + b_0c_2d_2 + b_0d_2 + b_0 + l_0 + m_0 + r_2 + r_3 && \rightarrow x'_2\\
f_3(a_1,b_0,c_1,d_0) &= b_0a_1c_1 + b_0c_1d_0 + b_0a_1 + a_1d_0 + b_0d_0 + m_0 + n_0 + r_3 + r_4 && \rightarrow x'_3\\
f_4(a_1,b_0,c_2,d_1) &= b_0a_1c_2 + b_0c_2d_1 + n_0 + k_1 + r_4 + r_5 && \rightarrow x'_4 \qquad \bigoplus_{i=0}^{8} x'_i = x_0\\
f_5(a_1,b_0,c_0,d_2) &= b_0a_1c_0 + b_0c_0d_2 + c_0a_1 + k_1 + l_1 + r_5 + r_6 && \rightarrow x'_5\\
f_6(a_2,b_0,c_0,d_1) &= b_0a_2c_0 + b_0c_0d_1 + b_0a_2 + c_0a_2 + l_1 + m_1 + r_6 + r_7 && \rightarrow x'_6\\
f_7(a_2,b_0,c_1,d_2) &= b_0a_2c_1 + b_0c_1d_2 + m_1 + n_1 + r_7 + r_8 && \rightarrow x'_7\\
f_8(a_2,b_0,c_2,d_0) &= b_0a_2c_2 + b_0c_2d_0 + a_2d_0 + a_2 + n_1 + r_8 + r_9 && \rightarrow x'_8
\end{aligned}$$

$$\begin{aligned}
f_9(a_0,b_1,c_2,d_0) &= b_1a_0c_2 + b_1c_2d_0 + b_1a_0 + b_1d_0 + b_1 + k_0 + r_9 + r_{10} && \rightarrow x'_9\\
f_{10}(a_0,b_1,c_0,d_1) &= b_1a_0c_0 + b_1c_0d_1 + a_0d_1 + b_1d_1 + k_0 + l_0 + r_{10} + r_{11} && \rightarrow x'_{10}\\
f_{11}(a_0,b_1,c_1,d_2) &= b_1a_0c_1 + b_1c_1d_2 + c_1a_0 + b_1d_2 + l_0 + m_0 + r_{11} + r_{12} && \rightarrow x'_{11}\\
f_{12}(a_1,b_1,c_2,d_2) &= b_1a_1c_2 + b_1c_2d_2 + b_1a_1 + m_0 + n_0 + r_{12} + r_{13} && \rightarrow x'_{12}\\
f_{13}(a_1,b_1,c_0,d_0) &= b_1a_1c_0 + b_1c_0d_0 + n_0 + k_1 + r_{13} + r_{14} && \rightarrow x'_{13} \qquad \bigoplus_{i=9}^{17} x'_i = x_1\\
f_{14}(a_1,b_1,c_1,d_1) &= b_1a_1c_1 + b_1c_1d_1 + c_1a_1 + a_1d_1 + c_1 + d_1 + k_1 + l_1 + r_{14} + r_{15} && \rightarrow x'_{14}\\
f_{15}(a_2,b_1,c_0,d_2) &= b_1a_2c_0 + b_1c_0d_2 + b_1a_2 + l_1 + m_1 + r_{15} + r_{16} && \rightarrow x'_{15}\\
f_{16}(a_2,b_1,c_1,d_0) &= b_1a_2c_1 + b_1c_1d_0 + c_1a_2 + m_1 + n_1 + r_{16} + r_{17} && \rightarrow x'_{16}\\
f_{17}(a_2,b_1,c_2,d_1) &= b_1a_2c_2 + b_1c_2d_1 + a_2d_1 + a_2 + n_1 + r_{17} + r_{18} && \rightarrow x'_{17}
\end{aligned}$$

$$\begin{aligned}
f_{18}(a_0,b_2,c_0,d_2) &= b_2a_0c_0 + b_2c_0d_2 + b_2a_0 + a_0d_2 + b_2d_2 + b_2 + a_0 + k_0 + r_{18} + r_{19} && \rightarrow x'_{18}\\
f_{19}(a_0,b_2,c_1,d_0) &= b_2a_0c_1 + b_2c_1d_0 + b_2d_0 + k_0 + l_0 + r_{19} + r_{20} && \rightarrow x'_{19}\\
f_{20}(a_0,b_2,c_2,d_1) &= b_2a_0c_2 + b_2c_2d_1 + c_2a_0 + b_2d_1 + l_0 + m_0 + r_{20} + r_{21} && \rightarrow x'_{20}\\
f_{21}(a_1,b_2,c_0,d_1) &= b_2a_1c_0 + b_2c_0d_1 + b_2a_1 + m_0 + n_0 + r_{21} + r_{22} && \rightarrow x'_{21}\\
f_{22}(a_1,b_2,c_2,d_0) &= b_2a_1c_2 + b_2c_2d_0 + c_2a_1 + n_0 + k_1 + r_{22} + r_{23} && \rightarrow x'_{22} \qquad \bigoplus_{i=18}^{26} x'_i = x_2\\
f_{23}(a_1,b_2,c_1,d_2) &= b_2a_1c_1 + b_2c_1d_2 + a_1d_2 + k_1 + l_1 + r_{23} + r_{24} && \rightarrow x'_{23}\\
f_{24}(a_2,b_2,c_1,d_1) &= b_2a_2c_1 + b_2c_1d_1 + b_2a_2 + l_1 + m_1 + r_{24} + r_{25} && \rightarrow x'_{24}\\
f_{25}(a_2,b_2,c_0,d_0) &= b_2a_2c_0 + b_2c_0d_0 + m_1 + n_1 + r_{25} + r_{26} && \rightarrow x'_{25}\\
f_{26}(a_2,b_2,c_2,d_2) &= b_2a_2c_2 + b_2c_2d_2 + c_2a_2 + a_2d_2 + c_2 + d_2 + n_1 + r_{26} + r_0 && \rightarrow x'_{26}
\end{aligned}$$

$$\begin{aligned}
h_0(b_0,c_0,d_0) &= b_0c_0 + d_0 + c_0 + b_0 + 1 + k_0 + l_0 + r_{27} + r_{28} && \rightarrow z'_0\\
h_1(b_0,c_1) &= b_0c_1 + l_0 + m_0 + r_{28} + r_{29} && \rightarrow z'_1 \qquad \bigoplus_{i=0}^{2} z'_i = z_0\\
h_2(b_0,c_2) &= b_0c_2 + m_0 + k_0 + r_{29} + r_{30} && \rightarrow z'_2
\end{aligned}$$

$$\begin{aligned}
h_3(b_1,c_0,d_1) &= b_1c_0 + d_1 + b_1 + k_0 + l_0 + r_{30} + r_{31} && \rightarrow z'_3\\
h_4(b_1,c_1) &= b_1c_1 + c_1 + l_0 + m_0 + r_{31} + r_{32} && \rightarrow z'_4 \qquad \bigoplus_{i=3}^{5} z'_i = z_1\\
h_5(b_1,c_2) &= b_1c_2 + m_0 + k_0 + r_{32} + r_{33} && \rightarrow z'_5
\end{aligned}$$

$$\begin{aligned}
h_6(b_2,c_0,d_2) &= b_2c_0 + d_2 + b_2 + k_0 + l_0 + r_{33} + r_{34} && \rightarrow z'_6\\
h_7(b_2,c_1) &= b_2c_1 + l_0 + m_0 + r_{34} + r_{35} && \rightarrow z'_7 \qquad \bigoplus_{i=6}^{8} z'_i = z_2\\
h_8(b_2,c_2) &= b_2c_2 + c_2 + m_0 + k_0 + r_{35} + r_{27} && \rightarrow z'_8
\end{aligned}$$

$$
\begin{array}{llll}
g_0(a_0,b_0,c_0,d_0) & = b_0c_0d_0 + b_0a_0 + b_0c_0 + b_0d_0 + c_0d_0 + k_0 + r_{36} + r_{37} & \to y'_0 & \\
g_1(a_0,b_0,c_1,d_1) & = b_0c_1d_1 + b_0c_1 + b_0d_1 + a_0 + k_0 + l_0 + r_{37} + r_{38} & \to y'_1 & \\
g_2(a_0,b_0,c_2,d_2) & = b_0c_2d_2 + b_0c_2 + b_0d_2 + c_2 + l_0 + m_0 + r_{38} + r_{39} & \to y'_2 & \\
g_3(a_1,b_0,c_1,d_0) & = b_0c_1d_0 + b_0a_1 + c_1d_0 + m_0 + n_0 + r_{39} + r_{40} & \to y'_3 & \\
g_4(a_1,b_0,c_2,d_1) & = b_0c_2d_1 + n_0 + k_1 + r_{40} + r_{41} & \to y'_4 & \bigoplus_{i=0}^{8} y'_i = y_0 \\
g_5(a_1,b_0,c_0,d_2) & = b_0c_0d_2 + c_0 + k_1 + l_1 + r_{41} + r_{42} & \to y'_5 & \\
g_6(a_2,b_0,c_0,d_1) & = b_0c_0d_1 + b_0a_2 + l_1 + m_1 + r_{42} + r_{43} & \to y'_6 & \\
g_7(a_2,b_0,c_1,d_2) & = b_0c_1d_2 + m_1 + n_1 + r_{43} + r_{44} & \to y'_7 & \\
g_8(a_2,b_0,c_2,d_0) & = b_0c_2d_0 + c_2d_0 + d_0 + n_1 + r_{44} + r_{45} & \to y'_8 & \\
\hline
g_9(a_0,b_1,c_2,d_0) & = b_1c_2d_0 + b_1a_0 + b_1c_2 + b_1d_0 + k_0 + r_{45} + r_{46} & \to y'_9 & \\
g_{10}(a_0,b_1,c_0,d_1) & = b_1c_0d_1 + b_1c_0 + b_1d_1 + c_0d_1 + d_1 + a_1 + k_0 + l_0 + r_{46} + r_{47} & \to y'_{10} & \\
g_{11}(a_0,b_1,c_1,d_2) & = b_1c_1d_2 + b_1c_1 + b_1d_2 + l_0 + m_0 + r_{47} + r_{48} & \to y'_{11} & \\
g_{12}(a_1,b_1,c_2,d_2) & = b_1c_2d_2 + b_1a_1 + m_0 + n_0 + r_{48} + r_{49} & \to y'_{12} & \\
g_{13}(a_1,b_1,c_0,d_0) & = b_1c_0d_0 + n_0 + k_1 + r_{49} + r_{50} & \to y'_{13} & \bigoplus_{i=9}^{17} y'_i = y_1 \\
g_{14}(a_1,b_1,c_1,d_1) & = b_1c_1d_1 + c_1d_1 + k_1 + l_1 + r_{50} + r_{51} & \to y'_{14} & \\
g_{15}(a_2,b_1,c_0,d_2) & = b_1c_0d_2 + b_1a_2 + l_1 + m_1 + r_{51} + r_{52} & \to y'_{15} & \\
g_{16}(a_2,b_1,c_1,d_0) & = b_1c_1d_0 + m_1 + n_1 + r_{52} + r_{53} & \to y'_{16} & \\
g_{17}(a_2,b_1,c_2,d_1) & = b_1c_2d_1 + c_2d_1 + c_2 + n_1 + r_{53} + r_{54} & \to y'_{17} & \\
\hline
g_{18}(a_0,b_2,c_0,d_2) & = b_2c_0d_2 + b_2a_0 + b_2c_0 + b_2d_2 + c_0d_2 + k_0 + r_{54} + r_{55} & \to y'_{18} & \\
g_{19}(a_0,b_2,c_1,d_0) & = b_2c_1d_0 + b_2c_1 + b_2d_0 + k_0 + l_0 + r_{55} + r_{56} & \to y'_{19} & \\
g_{20}(a_0,b_2,c_2,d_1) & = b_2c_2d_1 + b_2c_2 + b_2d_1 + l_0 + m_0 + r_{56} + r_{57} & \to y'_{20} & \\
g_{21}(a_1,b_2,c_0,d_1) & = b_2c_0d_1 + b_2a_1 + m_0 + n_0 + r_{57} + r_{58} & \to y'_{21} & \\
g_{22}(a_1,b_2,c_2,d_0) & = b_2c_2d_0 + n_0 + k_1 + r_{58} + r_{59} & \to y'_{22} & \bigoplus_{i=18}^{26} y'_i = y_2 \\
g_{23}(a_1,b_2,c_1,d_2) & = b_2c_1d_2 + c_1d_2 + k_1 + l_1 + r_{59} + r_{60} & \to y'_{23} & \\
g_{24}(a_2,b_2,c_1,d_1) & = b_2c_1d_1 + b_2a_2 + l_1 + m_1 + r_{60} + r_{61} & \to y'_{24} & \\
g_{25}(a_2,b_2,c_0,d_0) & = b_2c_0d_0 + c_0 + m_1 + n_1 + r_{61} + r_{62} & \to y'_{25} & \\
g_{26}(a_2,b_2,c_2,d_2) & = b_2c_2d_2 + c_2d_2 + d_2 + a_2 + n_1 + r_{62} + r_{36} & \to y'_{26} &
\end{array}
$$

$$
\begin{array}{llll}
u_0(a_0,c_0,d_0) & = c_0d_0 + d_0 + c_0 + a_0 + 1 + k_0 + l_0 + r_{63} + r_{64} & \to t'_0 & \\
u_1(c_0,d_1) & = c_0d_1 + l_0 + m_0 + r_{64} + r_{65} & \to t'_1 & \bigoplus_{i=0}^{2} t'_i = t_0 \\
u_2(c_0,d_2) & = c_0d_2 + m_0 + k_0 + r_{65} + r_{66} & \to t'_2 & \\
\hline
u_3(a_1,c_1,d_0) & = c_1d_0 + c_1 + a_1 + k_0 + l_0 + r_{66} + r_{67} & \to t'_3 & \\
u_4(c_1,d_1) & = c_1d_1 + d_1 + l_0 + m_0 + r_{67} + r_{68} & \to t'_4 & \bigoplus_{i=3}^{5} t'_i = t_1 \\
u_5(c_1,d_2) & = c_1d_2 + m_0 + k_0 + r_{68} + r_{69} & \to t'_5 & \\
\hline
u_6(a_2,c_2,d_0) & = c_2d_0 + c_2 + a_2 + k_0 + l_0 + r_{69} + r_{70} & \to t'_6 & \\
u_7(c_2,d_1) & = c_2d_1 + l_0 + m_0 + r_{70} + r_{71} & \to t'_7 & \bigoplus_{i=6}^{8} t'_i = t_2 \\
u_8(c_2,d_2) & = c_2d_2 + d_2 + m_0 + k_0 + r_{71} + r_{63} & \to t'_8 &
\end{array}
$$

C 3-share Prince S-Box/S-Box^{-1} with 108-bit Fresh Masks

$S(a,\ b,\ c,\ d) = (x,\ y,\ z,\ t)$ with lookup table `BF32AC916780E5D4`
$x = f(a,b,c,d) = 1 + ab + c + bc + abc + d + ad + cd$
$y = g(a,b,c,d) = 1 + ac + bc + abc + bd + bcd$
$z = h(a,b,c,d) = a + ab + d + ad + bd + abd + bcd$
$t = u(a,b,c,d) = 1 + b + bc + abc + d + abd + cd + acd$

$S^{-1}(a,\ b,\ c,\ d) = (x,\ y,\ z,\ t)$ with lookup table `B732FD89A6405EC1`
$x'' = f''(a,b,c,d) = 1 + ab + bc + d + abd + cd + acd$
$y'' = g''(a,b,c,d) = 1 + ac + bc + abc + bd + cd$
$z'' = h''(a,b,c,d) = a + ab + c + ac + bc + abc + bd + abd$
$t'' = u''(a,b,c,d) = 1 + a + b + ab + ac + bc + abc + cd + acd + bcd$

$q = MUX(sel,\ f_i(.),\ f_i''(.))$ selects either result of function $f_i(.)$ or $f_i''(.)$ based on the selctor signal `sel`.
We denote the uniform sharing of the functions in black.
$(k,\ l,\ m,\ n)$ are the input variables of the paired S-box, which are denoted in green.
Fresh masks are denoted in red.

The output shares are generated as follow:
$x_0 = \oplus_{i=0}^{8} q_i',\ x_1 = \oplus_{i=9}^{17} q_i',\ x_2 = \oplus_{i=18}^{26} q_i'$
$y_0 = \oplus_{i=27}^{38} q_i',\ y_1 = \oplus_{i=36}^{44} q_i',\ y_2 = \oplus_{i=45}^{53} q_i'$
$z_0 = \oplus_{i=54}^{62} q_i',\ z_1 = \oplus_{i=63}^{71} q_i',\ z_2 = \oplus_{i=72}^{80} q_i'$
$t_0 = \oplus_{i=81}^{95} q_i',\ t_1 = \oplus_{i=96}^{110} q_i',\ t_2 = \oplus_{i=111}^{125} q_i'$

The coordinate functions are given in full in the next pages.

$$\begin{array}{lcll}
f_0(a_0,b_0,c_0,d_0) & = & a_0b_0c_0 + a_0b_0 + c_0b_0 + c_0d_0 + a_0d_0 + c_0 + d_0 + b_0 + 1 & \\
f_0''(a_0,b_0,c_0,d_0) & = & a_0b_0d_0 + a_0c_0d_0 + a_0b_0 + c_0b_0 + c_0d_0 + d_0 + b_0 + 1 & \\
q_0 & = & MUX(sel, f_0(.), f_0''(.)) + k_0 + r_0 + r_1 & \rightarrow q_0' \\
f_1(a_0,b_0,c_1,d_1) & = & a_0b_0c_1 + c_1 + d_1 & \\
f_1''(a_0,b_0,c_1,d_1) & = & a_0b_0d_1 + a_0c_1d_1 & \\
q_1 & = & MUX(sel, f_1(.), f_1''(.)) + k_0 + l_0 + r_1 + r_2 & \rightarrow q_1' \\
f_2(a_0,b_0,c_2,d_2) & = & a_0b_0c_2 & \\
f_2''(a_0,b_0,c_2,d_2) & = & a_0b_0d_2 + a_0c_2d_2 & \\
q_2 & = & MUX(sel, f_2(.), f_2''(.)) + l_0 + m_0 + r_2 + r_3 & \rightarrow q_2' \\
f_3(a_0,b_1,c_1,d_0) & = & a_0b_1c_1 + a_0b_1 & \\
f_3''(a_0,b_1,c_1,d_0) & = & a_0b_1d_0 + a_0c_1d_0 + a_0b_1 + b_1 & \\
q_3 & = & MUX(sel, f_3(.), f_3''(.)) + m_0 + n_0 + r_3 + r_4 & \rightarrow q_3' \\
f_4(a_0,b_1,c_2,d_1) & = & a_0b_1c_2 & \\
f_4''(a_0,b_1,c_2,d_1) & = & a_0b_1d_1 + a_0c_2d_1 & \\
q_4 & = & MUX(sel, f_4(.), f_4''(.)) + n_0 + k_1 + r_4 + r_5 & \rightarrow q_4' \\
f_5(a_0,b_1,c_0,d_2) & = & a_0b_1c_0 + c_0b_1 + c_0d_2 + a_0d_2 & \\
f_5''(a_0,b_1,c_0,d_2) & = & a_0b_1d_2 + a_0c_0d_2 + c_0b_1 + c_0d_2 & \\
q_5 & = & MUX(sel, f_5(.), f_5''(.)) + k_1 + l_1 + r_5 + r_6 & \rightarrow q_5' \\
f_6(a_0,b_2,c_0,d_1) & = & a_0b_2c_0 + a_0b_2 + c_0b_2 + c_0d_1 + a_0d_1 & \\
f_6''(a_0,b_2,c_0,d_1) & = & a_0b_2d_1 + a_0c_0d_1 + a_0b_2 + c_0b_2 + c_0d_1 & \\
q_6 & = & MUX(sel, f_6(.), f_6''(.)) + l_1 + m_1 + r_6 + r_7 & \rightarrow q_6' \\
f_7(a_0,b_2,c_1,d_2) & = & a_0b_2c_1 & \\
f_7''(a_0,b_2,c_1,d_2) & = & a_0b_2d_2 + a_0c_1d_2 & \\
q_7 & = & MUX(sel, f_7(.), f_7''(.)) + m_1 + n_1 + r_7 + r_8 & \rightarrow q_7' \\
f_8(a_0,b_2,c_2,d_0) & = & a_0b_2c_2 & \\
f_8''(a_0,b_2,c_2,d_0) & = & a_0b_2d_0 + a_0c_2d_0 & \\
q_8 & = & MUX(sel, f_8(.), f_8''(.)) + n_1 + r_8 + r_9 & \rightarrow q_8'
\end{array}$$

$$
\begin{array}{lcll}
f_9(a_1,b_0,c_2,d_0) & = & a_1b_0c_2 + a_1b_0 + d_0 & \\
f''_9(a_1,b_0,c_2,d_0) & = & a_1b_0d_0 + a_1c_2d_0 + a_1b_0 & \\
q_9 & = & MUX(sel, f_9(.), f''_9(.)) + k_0 + r_9 + r_{10} & \rightarrow q'_9 \\
f_{10}(a_1,b_0,c_0,d_1) & = & a_1b_0c_0 + c_0 & \\
f''_{10}(a_1,b_0,c_0,d_1) & = & a_1b_0d_1 + a_1c_0d_1 + d_1 & \\
q_{10} & = & MUX(sel, f_{10}(.), f''_{10}(.)) + k_0 + l_0 + r_{10} + r_{11} & \rightarrow q'_{10} \\
f_{11}(a_1,b_0,c_1,d_2) & = & a_1b_0c_1 + c_1b_0 + c_1d_2 + a_1d_2 + c_1 & \\
f''_{11}(a_1,b_0,c_1,d_2) & = & a_1b_0d_2 + a_1c_1d_2 + c_1b_0 + c_1d_2 & \\
q_{11} & = & MUX(sel, f_{11}(.), f''_{11}(.)) + l_0 + m_0 + r_{11} + r_{12} & \rightarrow q'_{11} \\
f_{12}(a_1,b_1,c_2,d_2) & = & a_1b_1c_2 + a_1b_1 & \\
f''_{12}(a_1,b_1,c_2,d_2) & = & a_1b_1d_2 + a_1c_2d_2 + a_1b_1 + b_1 & \\
q_{12} & = & MUX(sel, f_{12}(.), f''_{12}(.)) + m_0 + n_0 + r_{12} + r_{13} & \rightarrow q'_{12} \\
f_{13}(a_1,b_1,c_0,d_0) & = & a_1b_1c_0 & \\
f''_{13}(a_1,b_1,c_0,d_0) & = & a_1b_1d_0 + a_1c_0d_0 & \\
q_{13} & = & MUX(sel, f_{13}(.), f''_{13}(.)) + n_0 + k_1 + r_{13} + r_{14} & \rightarrow q'_{13} \\
f_{14}(a_1,b_1,c_1,d_1) & = & a_1b_1c_1 + c_1b_1 + c_1d_1 + a_1d_1 & \\
f''_{14}(a_1,b_1,c_1,d_1) & = & a_1b_1d_1 + a_1c_1d_1 + c_1b_1 + c_1d_1 & \\
q_{14} & = & MUX(sel, f_{14}(.), f''_{14}(.)) + k_1 + l_1 + r_{14} + r_{15} & \rightarrow q'_{14} \\
f_{15}(a_1,b_2,c_0,d_2) & = & a_1b_2c_0 + a_1b_2 & \\
f''_{15}(a_1,b_2,c_0,d_2) & = & a_1b_2d_2 + a_1c_0d_2 + a_1b_2 & \\
q_{15} & = & MUX(sel, f_{15}(.), f''_{15}(.)) + l_1 + m_1 + r_{15} + r_{16} & \rightarrow q'_{15} \\
f_{16}(a_1,b_2,c_1,d_0) & = & a_1b_2c_1 + c_1b_2 + c_1d_0 + a_1d_0 & \\
f''_{16}(a_1,b_2,c_1,d_0) & = & a_1b_2d_0 + a_1c_1d_0 + c_1b_2 + c_1d_0 & \\
q_{16} & = & MUX(sel, f_{16}(.), f''_{16}(.)) + m_1 + n_1 + r_{16} + r_{17} & \rightarrow q'_{16} \\
f_{17}(a_1,b_2,c_2,d_1) & = & a_1b_2c_2 & \\
f''_{17}(a_1,b_2,c_2,d_1) & = & a_1b_2d_1 + a_1c_2d_1 + b_2 & \\
q_{17} & = & MUX(sel, f_{17}(.), f''_{17}(.)) + n_1 + r_{17} + r_{18} & \rightarrow q'_{17} \\
\hline
f_{18}(a_2,b_0,c_0,d_2) & = & a_2b_0c_0 + a_2b_0 + b_0 & \\
f''_{18}(a_2,b_0,c_0,d_2) & = & a_2b_0d_2 + a_2c_0d_2 + a_2b_0 + d_2 + b_0 & \\
q_{18} & = & MUX(sel, f_{18}(.), f''_{18}(.)) + k_0 + r_{18} + r_{19} & \rightarrow q'_{18} \\
f_{19}(a_2,b_0,c_1,d_0) & = & a_2b_0c_1 + c_1 + d_0 & \\
f''_{19}(a_2,b_0,c_1,d_0) & = & a_2b_0d_0 + a_2c_1d_0 & \\
q_{19} & = & MUX(sel, f_{19}(.), f''_{19}(.)) + k_0 + l_0 + r_{19} + r_{20} & \rightarrow q'_{19} \\
f_{20}(a_2,b_0,c_2,d_1) & = & a_2b_0c_2 + c_2b_0 + c_2d_1 + a_2d_1 & \\
f''_{20}(a_2,b_0,c_2,d_1) & = & a_2b_0d_1 + a_2c_2d_1 + c_2b_0 + c_2d_1 & \\
q_{20} & = & MUX(sel, f_{20}(.), f''_{20}(.)) + l_0 + m_0 + r_{20} + r_{21} & \rightarrow q'_{20} \\
f_{21}(a_2,b_1,c_0,d_1) & = & a_2b_1c_0 + a_2b_1 & \\
f''_{21}(a_2,b_1,c_0,d_1) & = & a_2b_1d_1 + a_2c_0d_1 + a_2b_1 & \\
q_{21} & = & MUX(sel, f_{21}(.), f''_{21}(.)) + m_0 + n_0 + r_{21} + r_{22} & \rightarrow q'_{21} \\
f_{22}(a_2,b_1,c_2,d_0) & = & a_2b_1c_2 + c_2b_1 + c_2d_0 + a_2d_0 & \\
f''_{22}(a_2,b_1,c_2,d_0) & = & a_2b_1d_0 + a_2c_2d_0 + c_2b_1 + c_2d_0 & \\
q_{22} & = & MUX(sel, f_{22}(.), f''_{22}(.)) + n_0 + k_1 + r_{22} + r_{23} & \rightarrow q'_{22} \\
f_{23}(a_2,b_1,c_1,d_2) & = & a_2b_1c_1 & \\
f''_{23}(a_2,b_1,c_1,d_2) & = & a_2b_1d_2 + a_2c_1d_2 & \\
q_{23} & = & MUX(sel, f_{23}(.), f''_{23}(.)) + k_1 + l_1 + r_{23} + r_{24} & \rightarrow q'_{23} \\
f_{24}(a_2,b_2,c_1,d_1) & = & a_2b_2c_1 + a_2b_2 & \\
f''_{24}(a_2,b_2,c_1,d_1) & = & a_2b_2d_1 + a_2c_1d_1 + a_2b_2 & \\
q_{24} & = & MUX(sel, f_{24}(.), f''_{24}(.)) + l_1 + m_1 + r_{24} + r_{25} & \rightarrow q'_{24} \\
f_{25}(a_2,b_2,c_0,d_0) & = & a_2b_2c_0 + c_0 & \\
f''_{25}(a_2,b_2,c_0,d_0) & = & a_2b_2d_0 + a_2c_0d_0 & \\
q_{25} & = & MUX(sel, f_{25}(.), f''_{25}(.)) + m_1 + n_1 + r_{25} + r_{26} & \rightarrow q'_{25} \\
f_{26}(a_2,b_2,c_2,d_2) & = & a_2b_2c_2 + c_2b_2 + c_2d_2 + a_2d_2 + c_2 + d_2 & \\
f''_{26}(a_2,b_2,c_2,d_2) & = & a_2b_2d_2 + a_2c_2d_2 + c_2b_2 + c_2d_2 + b_2 & \\
q_{26} & = & MUX(sel, f_{26}(.), f''_{26}(.)) + n_1 + r_{26} + r_0 & \rightarrow q'_{26}
\end{array}
$$

$$\begin{aligned}
g_0(a_0,b_0,c_0,d_0) &= b_0a_0c_0 + b_0d_0c_0 + d_0b_0 + c_0b_0 + a_0c_0 + 1 \\
g''_0(a_0,b_0,c_0,d_0) &= b_0a_0c_0 + c_0b_0 + a_0c_0 + d_0b_0 + d_0 + d_0c_0 + 1 \\
q_{27} &= MUX(sel, g_0(.), g''_0(.)) + k_0 + r_{27} + r_{28} && \rightarrow q'_{27} \\
g_1(a_0,b_0,c_1,d_1) &= b_0a_0c_1 + b_0d_1c_1 + d_1b_0 + c_1b_0 + c_1 \\
g''_1(a_0,b_0,c_1,d_1) &= b_0a_0c_1 + c_1b_0 + a_0c_1 + d_1b_0 + d_1 \\
q_{28} &= MUX(sel, g_1(.), g''_1(.)) + k_0 + l_0 + r_{28} + r_{29} && \rightarrow q'_{28} \\
g_2(a_0,b_0,c_2,d_2) &= b_0a_0c_2 + b_0d_2c_2 + d_2b_0 + c_2b_0 \\
g''_2(a_0,b_0,c_2,d_2) &= b_0a_0c_2 + c_2b_0 + a_0c_2 + d_2b_0 \\
q_{29} &= MUX(sel, g_2(.), g''_2(.)) + l_0 + m_0 + r_{29} + r_{30} && \rightarrow q'_{29} \\
g_3(a_1,b_0,c_1,d_0) &= b_0a_1c_1 + b_0d_0c_1 \\
g''_3(a_1,b_0,c_1,d_0) &= b_0a_1c_1 + d_0c_1 \\
q_{30} &= MUX(sel, g_3(.), g''_3(.)) + m_0 + n_0 + r_{30} + r_{31} && \rightarrow q'_{30} \\
g_4(a_1,b_0,c_2,d_1) &= b_0a_1c_2 + b_0d_1c_2 \\
g''_4(a_1,b_0,c_2,d_1) &= b_0a_1c_2 \\
q_{31} &= MUX(sel, g_4(.), g''_4(.)) + n_0 + k_1 + r_{31} + r_{32} && \rightarrow q'_{31} \\
g_5(a_1,b_0,c_0,d_2) &= b_0a_1c_0 + b_0d_2c_0 + a_1c_0 \\
g''_5(a_1,b_0,c_0,d_2) &= b_0a_1c_0 \\
q_{32} &= MUX(sel, g_5(.), g''_5(.)) + k_1 + l_1 + r_{32} + r_{33} && \rightarrow q'_{32} \\
g_6(a_2,b_0,c_0,d_1) &= b_0a_2c_0 + b_0d_1c_0 + a_2c_0 \\
g''_6(a_2,b_0,c_0,d_1) &= b_0a_2c_0 \\
q_{33} &= MUX(sel, g_6(.), g''_6(.)) + l_1 + m_1 + r_{33} + r_{34} && \rightarrow q'_{33} \\
g_7(a_2,b_0,c_1,d_2) &= b_0a_2c_1 + b_0d_2c_1 \\
g''_7(a_2,b_0,c_1,d_2) &= b_0a_2c_1 \\
q_{34} &= MUX(sel, g_7(.), g''_7(.)) + m_1 + n_1 + r_{34} + r_{35} && \rightarrow q'_{34} \\
g_8(a_2,b_0,c_2,d_0) &= b_0a_2c_2 + b_0d_0c_2 \\
g''_8(a_2,b_0,c_2,d_0) &= b_0a_2c_2 + d_0c_2 \\
q_{35} &= MUX(sel, g_8(.), g''_8(.)) + n_1 + r_{35} + r_{36} && \rightarrow q'_{35}
\end{aligned}$$

$$\begin{aligned}
g_9(a_0,b_1,c_2,d_0) &= b_1a_0c_2 + b_1d_0c_2 + b_1d_0 + b_1c_2 \\
g''_9(a_0,b_1,c_2,d_0) &= b_1a_0c_2 + b_1c_2 + b_1d_0 \\
q_{36} &= MUX(sel, g_9(.), g''_9(.)) + k_0 + r_{36} + r_{37} && \rightarrow q'_{36} \\
g_{10}(a_0,b_1,c_0,d_1) &= b_1a_0c_0 + b_1d_1c_0 + b_1d_1 + b_1c_0 + c_0 \\
g''_{10}(a_0,b_1,c_0,d_1) &= b_1a_0c_0 + b_1c_0 + b_1d_1 + d_1c_0 + d_1 \\
q_{37} &= MUX(sel, g_{10}(.), g''_{10}(.)) + k_0 + l_0 + r_{37} + r_{38} && \rightarrow q'_{37} \\
g_{11}(a_0,b_1,c_1,d_2) &= b_1a_0c_1 + b_1d_2c_1 + b_1d_2 + b_1c_1 + a_0c_1 + c_1 \\
g''_{11}(a_0,b_1,c_1,d_2) &= b_1a_0c_1 + b_1c_1 + b_1d_2 + d_2 \\
q_{38} &= MUX(sel, g_{11}(.), g''_{11}(.)) + l_0 + m_0 + r_{38} + r_{39} && \rightarrow q'_{38} \\
g_{12}(a_1,b_1,c_2,d_2) &= b_1a_1c_2 + b_1d_2c_2 \\
g''_{12}(a_1,b_1,c_2,d_2) &= b_1a_1c_2 + a_1c_2 \\
q_{39} &= MUX(sel, g_{12}(.), g''_{12}(.)) + m_0 + n_0 + r_{39} + r_{40} && \rightarrow q'_{39} \\
g_{13}(a_1,b_1,c_0,d_0) &= b_1a_1c_0 + b_1d_0c_0 \\
g''_{13}(a_1,b_1,c_0,d_0) &= b_1a_1c_0 + a_1c_0 \\
q_{40} &= MUX(sel, g_{13}(.), g''_{13}(.)) + n_0 + k_1 + r_{40} + r_{41} && \rightarrow q'_{40} \\
g_{14}(a_1,b_1,c_1,d_1) &= b_1a_1c_1 + b_1d_1c_1 + a_1c_1 \\
g''_{14}(a_1,b_1,c_1,d_1) &= b_1a_1c_1 + d_1c_1 + a_1c_1 \\
q_{41} &= MUX(sel, g_{14}(.), g''_{14}(.)) + k_1 + l_1 + r_{41} + r_{42} && \rightarrow q'_{41} \\
g_{15}(a_2,b_1,c_0,d_2) &= b_1a_2c_0 + b_1d_2c_0 \\
g''_{15}(a_2,b_1,c_0,d_2) &= b_1a_2c_0 \\
q_{42} &= MUX(sel, g_{15}(.), g''_{15}(.)) + l_1 + m_1 + r_{42} + r_{43} && \rightarrow q'_{42} \\
g_{16}(a_2,b_1,c_1,d_0) &= b_1a_2c_1 + b_1d_0c_1 + a_2c_1 \\
g''_{16}(a_2,b_1,c_1,d_0) &= b_1a_2c_1 \\
q_{43} &= MUX(sel, g_{16}(.), g''_{16}(.)) + m_1 + n_1 + r_{43} + r_{44} && \rightarrow q'_{43} \\
g_{17}(a_2,b_1,c_2,d_1) &= b_1a_2c_2 + b_1d_1c_2 \\
g''_{17}(a_2,b_1,c_2,d_1) &= b_1a_2c_2 \\
q_{44} &= MUX(sel, g_{17}(.), g''_{17}(.)) + n_1 + r_{44} + r_{45} && \rightarrow q'_{44}
\end{aligned}$$

$$
\begin{aligned}
g_{18}(a_0,b_2,c_0,d_2) &= b_2a_0c_0 + b_2d_2c_0 + b_2d_2 + c_0b_2 + c_0 \\
g''_{18}(a_0,b_2,c_0,d_2) &= b_2a_0c_0 + b_2c_0 + b_2d_2 + d_2c_0 + d_2 \\
q_{45} &= MUX(sel, g_{18}(.), g''_{18}(.)) + k_0 + r_{45} + r_{46} && \rightarrow q'_{45} \\
g_{19}(a_0,b_2,c_1,d_0) &= b_2a_0c_1 + b_2d_0c_1 + b_2d_0 + c_1b_2 \\
g''_{19}(a_0,b_2,c_1,d_0) &= b_2a_0c_1 + b_2c_1 + b_2d_0 + d_0 \\
q_{46} &= MUX(sel, g_{19}(.), g''_{19}(.)) + k_0 + l_0 + r_{46} + r_{47} && \rightarrow q'_{46} \\
g_{20}(a_0,b_2,c_2,d_1) &= b_2a_0c_2 + b_2d_1c_2 + b_2d_1 + c_2b_2 + a_0c_2 \\
g''_{20}(a_0,b_2,c_2,d_1) &= b_2a_0c_2 + b_2c_2 + b_2d_1 \\
q_{47} &= MUX(sel, g_{20}(.), g''_{20}(.)) + l_0 + m_0 + r_{47} + r_{48} && \rightarrow q'_{47} \\
g_{21}(a_1,b_2,c_0,d_1) &= b_2a_1c_0 + b_2d_1c_0 \\
g''_{21}(a_1,b_2,c_0,d_1) &= b_2a_1c_0 + d_2c_0 \\
q_{48} &= MUX(sel, g_{21}(.), g''_{21}(.)) + m_0 + n_0 + r_{48} + r_{49} && \rightarrow q'_{48} \\
g_{22}(a_1,b_2,c_2,d_0) &= b_2a_1c_2 + b_2d_0c_2 + a_1c_2 \\
g''_{22}(a_1,b_2,c_2,d_0) &= b_2a_1c_2 \\
q_{49} &= MUX(sel, g_{22}(.), g''_{22}(.)) + n_0 + k_1 + r_{49} + r_{50} && \rightarrow q'_{49} \\
g_{23}(a_1,b_2,c_1,d_2) &= b_2a_1c_1 + b_2d_2c_1 \\
g''_{23}(a_1,b_2,c_1,d_2) &= b_2a_1c_1 + d_2c_1 \\
q_{50} &= MUX(sel, g_{23}(.), g''_{23}(.)) + k_1 + l_1 + r_{50} + r_{51} && \rightarrow q'_{50} \\
g_{24}(a_2,b_2,c_1,d_1) &= b_2a_2c_1 + b_2d_1c_1 \\
g''_{24}(a_2,b_2,c_1,d_1) &= b_2a_2c_1 + a_2c_1 \\
q_{51} &= MUX(sel, g_{24}(.), g''_{24}(.)) + l_1 + m_1 + r_{51} + r_{52} && \rightarrow q'_{51} \\
g_{25}(a_2,b_2,c_0,d_0) &= b_2a_2c_0 + b_2d_0c_0 \\
g''_{25}(a_2,b_2,c_0,d_0) &= b_2a_2c_0 + a_2c_0 \\
q_{52} &= MUX(sel, g_{25}(.), g''_{25}(.)) + m_1 + n_1 + r_{52} + r_{53} && \rightarrow q'_{52} \\
g_{26}(a_2,b_2,c_2,d_2) &= b_2a_2c_2 + b_2d_2c_2 + a_2c_2 \\
g''_{26}(a_2,b_2,c_2,d_2) &= b_2a_2c_2 + a_2c_2 + d_2c_2 \\
q_{53} &= MUX(sel, g_{26}(.), g''_{26}(.)) + n_1 + r_{53} + r_{27} && \rightarrow q'_{53}
\end{aligned}
$$

$$
\begin{aligned}
h_0(a_0,b_0,c_0,d_0) &= b_0d_0c_0 + b_0a_0d_0 + a_0b_0 + b_0d_0 + a_0d_0 \\
h''_0(a_0,b_0,c_0,d_0) &= b_0a_0c_0 + b_0a_0d_0 + a_0c_0 + a_0b_0 + b_0c_0 \\
q_{54} &= MUX(sel, h_0(.), h''_0(.)) + k_0 + r_{54} + r_{55} && \rightarrow q'_{54} \\
h_1(a_0,b_0,c_1,d_1) &= b_0d_1c_1 + b_0a_0d_1 + b_0d_1 \\
h''_1(a_0,b_0,c_1,d_1) &= b_0a_0c_1 + b_0a_0d_1 + a_0c_1 + b_0c_1 + b_0d_1 \\
q_{55} &= MUX(sel, h_1(.), h''_1(.)) + k_0 + l_0 + r_{55} + r_{56} && \rightarrow q'_{55} \\
h_2(a_0,b_0,c_2,d_2) &= b_0d_2c_2 + b_0a_0d_2 + b_0d_2 + a_0 \\
h''_2(a_0,b_0,c_2,d_2) &= b_0a_0c_2 + b_0a_0d_2 + a_0c_2 + b_0c_2 + b_0d_2 + a_0 \\
q_{56} &= MUX(sel, h_2(.), h''_2(.)) + l_0 + m_0 + r_{56} + r_{57} && \rightarrow q'_{56} \\
h_3(a_1,b_0,c_1,d_0) &= b_0d_0c_1 + b_0a_1d_0 + a_1b_0 + a_1d_0 \\
h''_3(a_1,b_0,c_1,d_0) &= b_0a_1c_1 + b_0a_1d_0 + a_1b_0 \\
q_{57} &= MUX(sel, h_3(.), h''_3(.)) + m_0 + n_0 + r_{57} + r_{58} && \rightarrow q'_{57} \\
h_4(a_1,b_0,c_2,d_1) &= b_0d_1c_2 + b_0a_1d_1 + a_1 \\
h''_4(a_1,b_0,c_2,d_1) &= b_0a_1c_2 + b_0a_1d_1 \\
q_{58} &= MUX(sel, h_4(.), h''_4(.)) + n_0 + k_1 + r_{58} + r_{59} && \rightarrow q'_{58} \\
h_5(a_1,b_0,c_0,d_2) &= b_0d_2c_0 + b_0a_1d_2 \\
h''_5(a_1,b_0,c_0,d_2) &= b_0a_1c_0 + b_0a_1d_2 + c_0 \\
q_{59} &= MUX(sel, h_5(.), h''_5(.)) + k_1 + l_1 + r_{59} + r_{60} && \rightarrow q'_{59} \\
h_6(a_2,b_0,c_0,d_1) &= b_0d_1c_0 + b_0a_2d_1 + a_2b_0 \\
h''_6(a_2,b_0,c_0,d_1) &= b_0a_2c_0 + b_0a_2d_1 + a_2b_0 \\
q_{60} &= MUX(sel, h_6(.), h''_6(.)) + l_1 + m_1 + r_{60} + r_{61} && \rightarrow q'_{60} \\
h_7(a_2,b_0,c_1,d_2) &= b_0d_2c_1 + b_0a_2d_2 \\
h''_7(a_2,b_0,c_1,d_2) &= b_0a_2c_1 + b_0a_2d_2 \\
q_{61} &= MUX(sel, h_7(.), h''_7(.)) + m_1 + n_1 + r_{61} + r_{62} && \rightarrow q'_{61} \\
h_8(a_2,b_0,c_2,d_0) &= b_0d_0c_2 + b_0a_2d_0 + a_2d_0 + d_0 \\
h''_8(a_2,b_0,c_2,d_0) &= b_0a_2c_2 + b_0a_2d_0 + b_0d_0 \\
q_{62} &= MUX(sel, h_8(.), h''_8(.)) + n_1 + r_{62} + r_{63} && \rightarrow q'_{62}
\end{aligned}
$$

$$
\begin{array}{lll}
h_9(a_0,b_1,c_2,d_0) & = b_1d_0c_2 + b_1a_0d_0 + a_0b_1 + b_1d_0 & \\
h''_9(a_0,b_1,c_2,d_0) & = b_1a_0c_2 + b_1a_0d_0 + a_0b_1 + b_1c_2 + b_1d_0 & \\
q_{63} & = MUX(sel, h_9(.), h''_9(.)) + k_0 + r_{63} + r_{64} & \rightarrow q'_{63} \\
h_{10}(a_0,b_1,c_0,d_1) & = b_1d_1c_0 + b_1a_0d_1 + b_1d_1 + a_0d_1 + d_1 + a_0 & \\
h''_{10}(a_0,b_1,c_0,d_1) & = b_1a_0c_0 + b_1a_0d_1 + b_1c_0 + b_1d_1 & \\
q_{64} & = MUX(sel, h_{10}(.), h''_{10}(.)) + k_0 + l_0 + r_{64} + r_{65} & \rightarrow q'_{64} \\
h_{11}(a_0,b_1,c_1,d_2) & = b_1d_2c_1 + b_1a_0d_2 + b_1d_2 & \\
h''_{11}(a_0,b_1,c_1,d_2) & = b_1a_0c_1 + b_1a_0d_2 + b_1c_1 + b_1d_2 & \\
q_{65} & = MUX(sel, h_{11}(.), h''_{11}(.)) + l_0 + m_0 + r_{65} + r_{66} & \rightarrow q'_{65} \\
h_{12}(a_1,b_1,c_2,d_2) & = b_1d_2c_2 + b_1a_1d_2 + a_1b_1 & \\
h''_{12}(a_1,b_1,c_2,d_2) & = b_1a_1c_2 + b_1a_1d_2 + a_1c_0 + a_1b_1 & \\
q_{66} & = MUX(sel, h_{12}(.), h''_{12}(.)) + m_0 + n_0 + r_{66} + r_{67} & \rightarrow q'_{66} \\
h_{13}(a_1,b_1,c_0,d_0) & = b_1d_0c_0 + b_1a_1d_0 & \\
h''_{13}(a_1,b_1,c_0,d_0) & = b_1a_1c_0 + b_1a_1d_0 + a_1c_1 & \\
q_{67} & = MUX(sel, h_{13}(.), h''_{13}(.)) + n_0 + k_1 + r_{67} + r_{68} & \rightarrow q'_{67} \\
h_{14}(a_1,b_1,c_1,d_1) & = b_1d_1c_1 + b_1a_1d_1 + a_1d_1 & \\
h''_{14}(a_1,b_1,c_1,d_1) & = b_1a_1c_1 + b_1a_1d_1 + a_1c_2 + a_1 + c_1 & \\
q_{68} & = MUX(sel, h_{14}(.), h''_{14}(.)) + k_1 + l_1 + r_{68} + r_{69} & \rightarrow q'_{68} \\
h_{15}(a_2,b_1,c_0,d_2) & = b_1d_2c_0 + b_1a_2d_2 + a_2b_1 & \\
h''_{15}(a_2,b_1,c_0,d_2) & = b_1a_2c_0 + b_1a_2d_2 + a_2b_1 & \\
q_{69} & = MUX(sel, h_{15}(.), h''_{15}(.)) + l_1 + m_1 + r_{69} + r_{70} & \rightarrow q'_{69} \\
h_{16}(a_2,b_1,c_1,d_0) & = b_1d_0c_1 + b_1a_2d_0 & \\
h''_{16}(a_2,b_1,c_1,d_0) & = b_1a_2c_1 + b_1a_2d_0 & \\
q_{70} & = MUX(sel, h_{16}(.), h''_{16}(.)) + m_1 + n_1 + r_{70} + r_{71} & \rightarrow q'_{70} \\
h_{17}(a_2,b_1,c_2,d_1) & = b_1d_1c_2 + b_1a_2d_1 + a_2d_1 & \\
h''_{17}(a_2,b_1,c_2,d_1) & = b_1a_2c_2 + b_1a_2d_1 & \\
q_{71} & = MUX(sel, h_{17}(.), h''_{17}(.)) + n_1 + r_{71} + r_{72} & \rightarrow q'_{71} \\
\hline
h_{18}(a_0,b_2,c_0,d_2) & = b_2d_2c_0 + b_2a_0d_2 + a_0b_2 + b_2d_2 + a_0d_2 + a_0 & \\
h''_{18}(a_0,b_2,c_0,d_2) & = b_2a_0c_0 + b_2a_0d_2 + a_0b_2 + b_2c_0 + b_2d_2 & \\
q_{72} & = MUX(sel, h_{18}(.), h''_{18}(.)) + k_0 + r_{72} + r_{73} & \rightarrow q'_{72} \\
h_{19}(a_0,b_2,c_1,d_0) & = b_2d_0c_1 + b_2a_0d_0 + b_2d_0 & \\
h''_{19}(a_0,b_2,c_1,d_0) & = b_2a_0c_1 + b_2a_0d_0 + b_2c_1 + b_2d_0 & \\
q_{73} & = MUX(sel, h_{19}(.), h''_{19}(.)) + k_0 + l_0 + r_{73} + r_{74} & \rightarrow q'_{73} \\
h_{20}(a_0,b_2,c_2,d_1) & = b_2d_1c_2 + b_2a_0d_1 + b_2d_1 & \\
h''_{20}(a_0,b_2,c_2,d_1) & = b_2a_0c_2 + b_2a_0d_1 + b_2c_2 + b_2d_1 & \\
q_{74} & = MUX(sel, h_{20}(.), h''_{20}(.)) + l_0 + m_0 + r_{74} + r_{75} & \rightarrow q'_{74} \\
h_{21}(a_1,b_2,c_0,d_1) & = b_2d_1c_0 + b_2a_1d_1 + a_1b_2 & \\
h''_{21}(a_1,b_2,c_0,d_1) & = b_2a_1c_0 + b_2a_1d_1 + a_1b_2 & \\
q_{75} & = MUX(sel, h_{21}(.), h''_{21}(.)) + m_0 + n_0 + r_{75} + r_{76} & \rightarrow q'_{75} \\
h_{22}(a_1,b_2,c_2,d_0) & = b_2d_0c_2 + b_2a_1d_0 & \\
h''_{22}(a_1,b_2,c_2,d_0) & = b_2a_1c_2 + b_2a_1d_0 & \\
q_{76} & = MUX(sel, h_{22}(.), h''_{22}(.)) + n_0 + k_1 + r_{76} + r_{77} & \rightarrow q'_{76} \\
h_{23}(a_1,b_2,c_1,d_2) & = b_2d_2c_1 + b_2a_1d_2 + a_1d_2 & \\
h''_{23}(a_1,b_2,c_1,d_2) & = b_2a_1c_1 + b_2a_1d_2 & \\
q_{77} & = MUX(sel, h_{23}(.), h''_{23}(.)) + k_1 + l_1 + r_{77} + r_{78} & \rightarrow q'_{77} \\
h_{24}(a_2,b_2,c_1,d_1) & = b_2d_1c_1 + b_2a_2d_1 + a_2b_2 & \\
h''_{24}(a_2,b_2,c_1,d_1) & = b_2a_2c_1 + b_2a_2d_1 + a_2c_0 + a_2b_2 & \\
q_{78} & = MUX(sel, h_{24}(.), h''_{24}(.)) + l_1 + m_1 + r_{78} + r_{79} & \rightarrow q'_{78} \\
h_{25}(a_2,b_2,c_0,d_0) & = b_2d_0c_0 + b_2a_2d_0 & \\
h''_{25}(a_2,b_2,c_0,d_0) & = b_2a_2c_0 + b_2a_2d_0 + a_2c_1 & \\
q_{79} & = MUX(sel, h_{25}(.), h''_{25}(.)) + m_1 + n_1 + r_{79} + r_{80} & \rightarrow q'_{79} \\
h_{26}(a_2,b_2,c_2,d_2) & = b_2d_2c_2 + b_2a_2d_2 + a_2d_2 + a_2 + d_2 & \\
h''_{26}(a_2,b_2,c_2,d_2) & = b_2a_2c_2 + b_2a_2d_2 + a_2c_2 + a_2 + c_2 & \\
q_{80} & = MUX(sel, h_{26}(.), h''_{26}(.)) + n_1 + r_{80} + r_{54} & \rightarrow q'_{80} \\
\hdashline
\end{array}
$$

$$
\begin{aligned}
u_0(a_0,b_0,c_0,d_0) &= a_0c_0b_0 + a_0c_0d_0 + a_0b_0d_0 + c_0b_0 + c_0d_0 + b_0 \\
u''_0(a_0,b_0,c_0,d_0) &= c_0a_0b_0 + c_0a_0d_0 + c_0b_0d_0 + a_0b_0 + c_0b_0 + c_0d_0 + a_0 + b_0 \\
q_{81} &= MUX(sel, u_{81}(.), u''_{81}(.)) + k_0 + r_{54} + r_{55} && \rightarrow q'_{81} \\
u_1(a_0,b_1,c_0,d_1) &= a_0c_0b_1 + a_0c_0d_1 + a_0b_1d_1 + c_0b_1 + 1 \\
u''_1(a_0,b_1,c_0,d_1) &= c_0a_0b_1 + c_0a_0d_1 + c_0b_1d_1 + a_0b_1 + c_0b_1 + c_0d_1 + b_1 \\
q_{82} &= MUX(sel, u_{82}(.), u''_{82}(.)) + k_0 + l_0 + r_{55} + r_{56} && \rightarrow q'_{82} \\
u_2(a_0,b_2,c_0,d_2) &= a_0c_0b_2 + a_0c_0d_2 + a_0b_2d_2 + c_0b_2 \\
u''_2(a_0,b_2,c_0,d_2) &= c_0a_0b_2 + c_0a_0d_2 + c_0b_2d_2 + a_0b_2 + c_0a_0 + c_0b_2 + c_0d_2 + d_2 + b_2 \\
q_{83} &= MUX(sel, u_{83}(.), u''_{83}(.)) + l_0 + m_0 + r_{56} + r_{57} && \rightarrow q'_{83} \\
u_3(a_1,b_1,c_0,d_0) &= 0 \\
u''_3(a_1,b_1,c_0,d_0) &= c_0a_1b_1 + c_0a_1d_0 + c_0b_1d_0 + c_0a_1 + a_1 + d_0 \\
q_{84} &= MUX(sel, u_{84}(.), u''_{84}(.)) + m_0 + n_0 + r_{57} + r_{58} && \rightarrow q'_{84} \\
u_4(a_1,b_2,c_0,d_1) &= 0 \\
u''_4(a_1,b_2,c_0,d_1) &= c_0a_1b_2 + c_0a_1d_1 + c_0b_2d_1 + 1 \\
q_{85} &= MUX(sel, u_{85}(.), u''_{85}(.)) + n_0 + k_1 + r_{58} + r_{59} && \rightarrow q'_{85} \\
u_5(a_1,b_0,c_0,d_2) &= 0 \\
u''_5(a_1,b_0,c_0,d_2) &= c_0a_1b_0 + c_0a_1d_2 + c_0b_0d_2 \\
q_{86} &= MUX(sel, u_{86}(.), u''_{86}(.)) + k_1 + l_1 + r_{59} + r_{60} && \rightarrow q'_{86} \\
u_6(a_2,b_0,c_0,d_1) &= 0 \\
u''_6(a_2,b_0,c_0,d_1) &= c_0a_2b_0 + c_0a_2d_1 + c_0b_0d_1 \\
q_{87} &= MUX(sel, u_{87}(.), u''_{87}(.)) + l_1 + m_1 + r_{60} + r_{61} && \rightarrow q'_{87} \\
u_7(a_2,b_1,c_0,d_2) &= 0 \\
u''_7(a_2,b_1,c_0,d_2) &= c_0a_2b_1 + c_0a_2d_2 + c_0b_1d_2 \\
q_{88} &= MUX(sel, u_{88}(.), u''_{88}(.)) + m_1 + n_1 + r_{61} + r_{62} && \rightarrow q'_{88} \\
u_8(a_2,b_2,c_0,d_0) &= 0 \\
u''_8(a_2,b_2,c_0,d_0) &= c_0a_2b_2 + c_0a_2d_0 + c_0b_2d_0 + c_0a_2 \\
q_{89} &= MUX(sel, u_{89}(.), u''_{89}(.)) + n_1 + r_{62} + r_{63} && \rightarrow q'_{89} \\
u_9(a_0,b_1,c_1,d_0) &= a_0c_1b_1 + a_0c_1d_0 + a_0b_1d_0 + c_1d_0 \\
u''_9(a_0,b_1,c_1,d_0) &= 0 \\
q_{90} &= MUX(sel, u_{90}(.), u''_{90}(.)) + m_0 + n_0 + r_{57} + r_{58} && \rightarrow q'_{90} \\
u_{10}(a_0,b_2,c_1,d_1) &= a_0c_1b_2 + a_0c_1d_1 + a_0b_2d_1 \\
u''_{10}(a_0,b_2,c_1,d_1) &= 0 \\
q_{91} &= MUX(sel, u_{91}(.), u''_{91}(.)) + n_0 + k_1 + r_{58} + r_{59} && \rightarrow q'_{91} \\
u_{11}(a_0,b_0,c_1,d_2) &= a_0c_1b_0 + a_0c_1d_2 + a_0b_0d_2 \\
u''_{11}(a_0,b_0,c_1,d_2) &= 0 \\
q_{92} &= MUX(sel, u_{92}(.), u''_{92}(.)) + k_1 + l_1 + r_{59} + r_{60} && \rightarrow q'_{92} \\
u_{12}(a_0,b_0,c_2,d_1) &= a_0c_2b_0 + a_0c_2d_1 + a_0b_0d_1 \\
u''_{12}(a_0,b_0,c_2,d_1) &= 0 \\
q_{93} &= MUX(sel, u_{93}(.), u''_{93}(.)) + l_1 + m_1 + r_{60} + r_{61} && \rightarrow q'_{93} \\
u_{13}(a_0,b_1,c_2,d_2) &= a_0c_2b_1 + a_0c_2d_2 + a_0b_1d_2 \\
u''_{13}(a_0,b_1,c_2,d_2) &= 0 \\
q_{94} &= MUX(sel, u_{94}(.), u''_{94}(.)) + m_1 + n_1 + r_{61} + r_{62} && \rightarrow q'_{94} \\
u_{14}(a_0,b_2,c_2,d_0) &= a_0c_2b_2 + a_0c_2d_0 + a_0b_2d_0 + c_2d_0 \\
u''_{14}(a_0,b_2,c_2,d_0) &= 0 \\
q_{95} &= MUX(sel, u_{95}(.), u''_{95}(.)) + n_1 + r_{62} + r_{63} && \rightarrow q'_{95}
\end{aligned}
$$

$$
\begin{aligned}
u_{15}(a_0,b_2,c_1,d_0) &= 0 \\
u''_{15}(a_0,b_2,c_1,d_0) &= c_1a_0b_2 + c_1a_0d_0 + c_1b_2d_0 + c_1d_0 \\
q_{96} &= MUX(sel, u_{15}(.), u''_{15}(.)) + k_0 + r_{63} + r_{64} && \rightarrow q'_{96} \\
u_{16}(a_0,b_0,c_1,d_1) &= 0 \\
u''_{16}(a_0,b_0,c_1,d_1) &= c_1a_0b_0 + c_1a_0d_1 + c_1b_0d_1 + c_1d_1 \\
q_{97} &= MUX(sel, u_{16}(.), u''_{16}(.)) + k_0 + l_0 + r_{64} + r_{65} && \rightarrow q'_{97} \\
u_{17}(a_0,b_1,c_1,d_2) &= 0 \\
u''_{17}(a_0,b_1,c_1,d_2) &= c_1a_0b_1 + c_1a_0d_2 + c_1b_1d_2 + c_1a_0 + c_1d_2 + a_0 + d_2 \\
q_{98} &= MUX(sel, u_{17}(.), u''_{17}(.)) + l_0 + m_0 + r_{65} + r_{66} && \rightarrow q'_{98} \\
u_{18}(a_1,b_2,c_1,d_2) &= a_1c_1b_2 + a_1c_1d_2 + a_1b_2d_2 + c_1b_2 \\
u''_{18}(a_1,b_2,c_1,d_2) &= c_1a_1b_2 + c_1a_1d_2 + c_1b_2d_2 + a_1b_2 + c_1a_1 + c_1b_2 \\
q_{99} &= MUX(sel, u_{18}(.), u''_{18}(.)) + m_0 + n_0 + r_{66} + r_{67} && \rightarrow q'_{99} \\
u_{19}(a_1,b_0,c_1,d_0) &= a_1c_1b_0 + a_1c_1d_0 + a_1b_0d_0 + c_1b_0 \\
u''_{19}(a_1,b_0,c_1,d_0) &= c_1a_1b_0 + c_1a_1d_0 + c_1b_0d_0 + a_1b_0 + c_1b_0 \\
q_{100} &= MUX(sel, u_{19}(.), u''_{19}(.)) + n_0 + k_1 + r_{67} + r_{68} && \rightarrow q'_{100} \\
u_{20}(a_1,b_1,c_1,d_1) &= a_1c_1b_1 + a_1c_1d_1 + a_1b_1d_1 + c_1b_1 + c_1d_1 + b_1 + d_1 \\
u''_{20}(a_1,b_1,c_1,d_1) &= c_1a_1b_1 + c_1a_1d_1 + c_1b_1d_1 + a_1b_1 + c_1b_1 + b_1 + b_1 \\
q_{101} &= MUX(sel, u_{20}(.), u''_{20}(.)) + k_1 + l_1 + r_{68} + r_{69} && \rightarrow q'_{101}
\end{aligned}
$$

$$
\begin{array}{lcll}
u_{21}(a_2,b_0,c_1,d_2) & = & 0 & \\
u''_{21}(a_2,b_0,c_1,d_2) & = & c_1a_2b_0 + c_1a_2d_2 + c_1b_0d_2 & \\
q_{102} & = & MUX(sel, u_{21}(.), u''_{21}(.)) + l_1 + m_1 + r_{69} + r_{70} & \rightarrow q'_{102} \\
u_{22}(a_2,b_1,c_1,d_0) & = & 0 & \\
u''_{22}(a_2,b_1,c_1,d_0) & = & c_1a_2b_1 + c_1a_2d_0 + c_1b_1d_0 & \\
q_{103} & = & MUX(sel, u_{22}(.), u''_{22}(.)) + m_1 + n_1 + r_{70} + r_{71} & \rightarrow q'_{103} \\
u_{23}(a_2,b_2,c_1,d_1) & = & 0 & \\
u''_{23}(a_2,b_2,c_1,d_1) & = & c_1a_2b_2 + c_1a_2d_1 + c_1b_2d_1 + c_1a_2 & \\
q_{104} & = & MUX(sel, u_{23}(.), u''_{23}(.)) + n_1 + r_{71} + r_{72} & \rightarrow q'_{104} \\
u_{24}(a_1,b_2,c_0,d_0) & = & a_1c_0b_2 + a_1c_0d_0 + a_1b_2d_0 & \\
u''_{24}(a_1,b_2,c_0,d_0) & = & 0 & \\
q_{105} & = & MUX(sel, u_{24}(.), u''_{24}(.)) + m_0 + n_0 + r_{66} + r_{67} & \rightarrow q'_{105} \\
u_{25}(a_1,b_0,c_0,d_1) & = & a_1c_0b_0 + a_1c_0d_1 + a_1b_0d_1 + c_0d_1 & \\
u''_{25}(a_1,b_0,c_0,d_1) & = & 0 & \\
q_{106} & = & MUX(sel, u_{25}(.), u''_{25}(.)) + n_0 + k_1 + r_{67} + r_{68} & \rightarrow q'_{106} \\
u_{26}(a_1,b_1,c_0,d_2) & = & a_1c_0b_1 + a_1c_0d_2 + a_1b_1d_2 & \\
u''_{26}(a_1,b_1,c_0,d_2) & = & 0 & \\
q_{107} & = & MUX(sel, u_{26}(.), u''_{26}(.)) + k_1 + l_1 + r_{68} + r_{69} & \rightarrow q'_{107} \\
u_{27}(a_1,b_0,c_2,d_2) & = & a_1c_2b_0 + a_1c_2d_2 + a_1b_0d_2 & \\
u''_{27}(a_1,b_0,c_2,d_2) & = & 0 & \\
q_{108} & = & MUX(sel, u_{27}(.), u''_{27}(.)) + l_1 + m_1 + r_{69} + r_{70} & \rightarrow q'_{108} \\
u_{28}(a_1,b_1,c_2,d_0) & = & a_1c_2b_1 + a_1c_2d_0 + a_1b_1d_0 & \\
u''_{28}(a_1,b_1,c_2,d_0) & = & 0 & \\
q_{109} & = & MUX(sel, u_{28}(.), u''_{28}(.)) + m_1 + n_1 + r_{70} + r_{71} & \rightarrow q'_{109} \\
u_{29}(a_1,b_2,c_2,d_1) & = & a_1c_2b_2 + a_1c_2d_1 + a_1b_2d_1 + c_2d_1 & \\
u''_{29}(a_1,b_2,c_2,d_1) & = & 0 & \\
q_{110} & = & MUX(sel, u_{29}(.), u''_{29}(.)) + n_1 + r_{71} + r_{72} & \rightarrow q'_{110} \\
\hline
u_{30}(a_0,b_0,c_2,d_2) & = & 0 & \\
u''_{30}(a_0,b_0,c_2,d_2) & = & c_2a_0b_0 + c_2a_0d_2 + c_2b_0d_2 + c_2d_2 & \\
q_{111} & = & MUX(sel, u_{30}(.), u''_{30}(.)) + k_0 + r_{72} + r_{73} & \rightarrow q'_{111} \\
u_{31}(a_0,b_1,c_2,d_0) & = & 0 & \\
u''_{31}(a_0,b_1,c_2,d_0) & = & c_2a_0b_1 + c_2a_0d_0 + c_2b_1d_0 + c_2d_0 + a_0 + d_0 & \\
q_{112} & = & MUX(sel, u_{31}(.), u''_{31}(.)) + k_0 + l_0 + r_{73} + r_{74} & \rightarrow q'_{112} \\
u_{32}(a_0,b_2,c_2,d_1) & = & 0 & \\
u''_{32}(a_0,b_2,c_2,d_1) & = & c_2a_0b_2 + c_2a_0d_1 + c_2b_2d_1 + c_2a_0 + c_2d_1 & \\
q_{113} & = & MUX(sel, u_{32}(.), u''_{32}(.)) + l_0 + m_0 + r_{74} + r_{75} & \rightarrow q'_{113} \\
u_{33}(a_1,b_0,c_2,d_1) & = & 0 & \\
u''_{33}(a_1,b_0,c_2,d_1) & = & c_2a_1b_0 + c_2a_1d_1 + c_2b_0d_1 + c_2a_1 & \\
q_{114} & = & MUX(sel, u_{33}(.), u''_{33}(.)) + m_0 + n_0 + r_{75} + r_{76} & \rightarrow q'_{114} \\
u_{34}(a_1,b_2,c_2,d_0) & = & 0 & \\
u''_{34}(a_1,b_2,c_2,d_0) & = & c_2a_1b_2 + c_2a_1d_0 + c_2b_2d_0 & \\
q_{115} & = & MUX(sel, u_{34}(.), u''_{34}(.)) + n_0 + k_1 + r_{76} + r_{77} & \rightarrow q'_{115} \\
u_{35}(a_1,b_1,c_2,d_2) & = & 0 & \\
u''_{35}(a_1,b_1,c_2,d_2) & = & c_2a_1b_1 + c_2a_1d_2 + c_2b_1d_2 & \\
q_{116} & = & MUX(sel, u_{35}(.), u''_{35}(.)) + k_1 + l_1 + r_{77} + r_{78} & \rightarrow q'_{116} \\
u_{36}(a_2,b_1,c_2,d_1) & = & a_2c_2b_1 + a_2c_2d_1 + a_2b_1d_1 + c_2b_1 & \\
u''_{36}(a_2,b_1,c_2,d_1) & = & c_2a_2b_1 + c_2a_2d_1 + c_2b_1d_1 + a_2b_1 + c_2b_1 & \\
q_{117} & = & MUX(sel, u_{36}(.), u''_{36}(.)) + l_1 + m_1 + r_{78} + r_{79} & \rightarrow q'_{117} \\
u_{37}(a_2,b_0,c_2,d_0) & = & a_2c_2b_0 + a_2c_2d_0 + a_2b_0d_0 + c_2b_0 & \\
u''_{37}(a_2,b_0,c_2,d_0) & = & c_2a_2b_0 + c_2a_2d_0 + c_2b_0d_0 + a_2b_0 + c_2b_0 & \\
q_{118} & = & MUX(sel, u_{37}(.), u''_{37}(.)) + m_1 + n_1 + r_{79} + r_{80} & \rightarrow q'_{118} \\
u_{38}(a_2,b_2,c_2,d_2) & = & a_2c_2b_2 + a_2c_2d_2 + a_2b_2d_2 + c_2b_2 + c_2d_2 + b_2 + d_2 & \\
u''_{38}(a_2,b_2,c_2,d_2) & = & c_2a_2b_2 + c_2a_2d_2 + c_2b_2d_2 + a_2b_2 + c_2a_2 + c_2b_2 + a_2 & \\
q_{119} & = & MUX(sel, u_{38}(.), u''_{38}(.)) + n_1 + r_{80} + r_{54} & \rightarrow q'_{119} \\
u_{39}(a_2,b_0,c_1,d_1) & = & a_2c_1b_0 + a_2c_1d_1 + a_2b_0d_1 & \\
u''_{39}(a_2,b_0,c_1,d_1) & = & 0 & \\
q_{120} & = & MUX(sel, u_{39}(.), u''_{39}(.)) + m_0 + n_0 + r_{75} + r_{76} & \rightarrow q'_{120} \\
u_{40}(a_2,b_2,c_1,d_0) & = & a_2c_1b_2 + a_2c_1d_0 + a_2b_2d_0 & \\
u''_{40}(a_2,b_2,c_1,d_0) & = & 0 & \\
q_{121} & = & MUX(sel, u_{40}(.), u''_{40}(.)) + n_0 + k_1 + r_{76} + r_{77} & \rightarrow q'_{121} \\
u_{41}(a_2,b_1,c_1,d_2) & = & a_2c_1b_1 + a_2c_1d_2 + a_2b_1d_2 + c_1d_2 & \\
u''_{41}(a_2,b_1,c_1,d_2) & = & 0 & \\
q_{122} & = & MUX(sel, u_{41}(.), u''_{41}(.)) + k_1 + l_1 + r_{77} + r_{78} & \rightarrow q'_{122} \\
u_{42}(a_2,b_0,c_0,d_2) & = & a_2c_0b_0 + a_2c_0d_2 + a_2b_0d_2 + c_0d_2 & \\
u''_{42}(a_2,b_0,c_0,d_2) & = & 0 & \\
q_{123} & = & MUX(sel, u_{42}(.), u''_{42}(.)) + l_1 + m_1 + r_{78} + r_{79} & \rightarrow q'_{123} \\
u_{43}(a_2,b_1,c_0,d_0) & = & a_2c_0b_1 + a_2c_0d_0 + a_2b_1d_0 + d_0 & \\
u''_{43}(a_2,b_1,c_0,d_0) & = & 0 & \\
q_{124} & = & MUX(sel, u_{43}(.), u''_{43}(.)) + m_1 + n_1 + r_{79} + r_{80} & \rightarrow q'_{124} \\
u_{44}(a_2,b_2,c_0,d_1) & = & a_2c_0b_2 + a_2c_0d_1 + a_2b_2d_1 & \\
u''_{44}(a_2,b_2,c_0,d_1) & = & 0 & \\
q_{125} & = & MUX(sel, u_{44}(.), u''_{44}(.)) + n_1 + r_{80} + r_{54} & \rightarrow q'_{125}
\end{array}
$$

D Result of Experimental Analyses on Our Midori and SKINNY Implementations

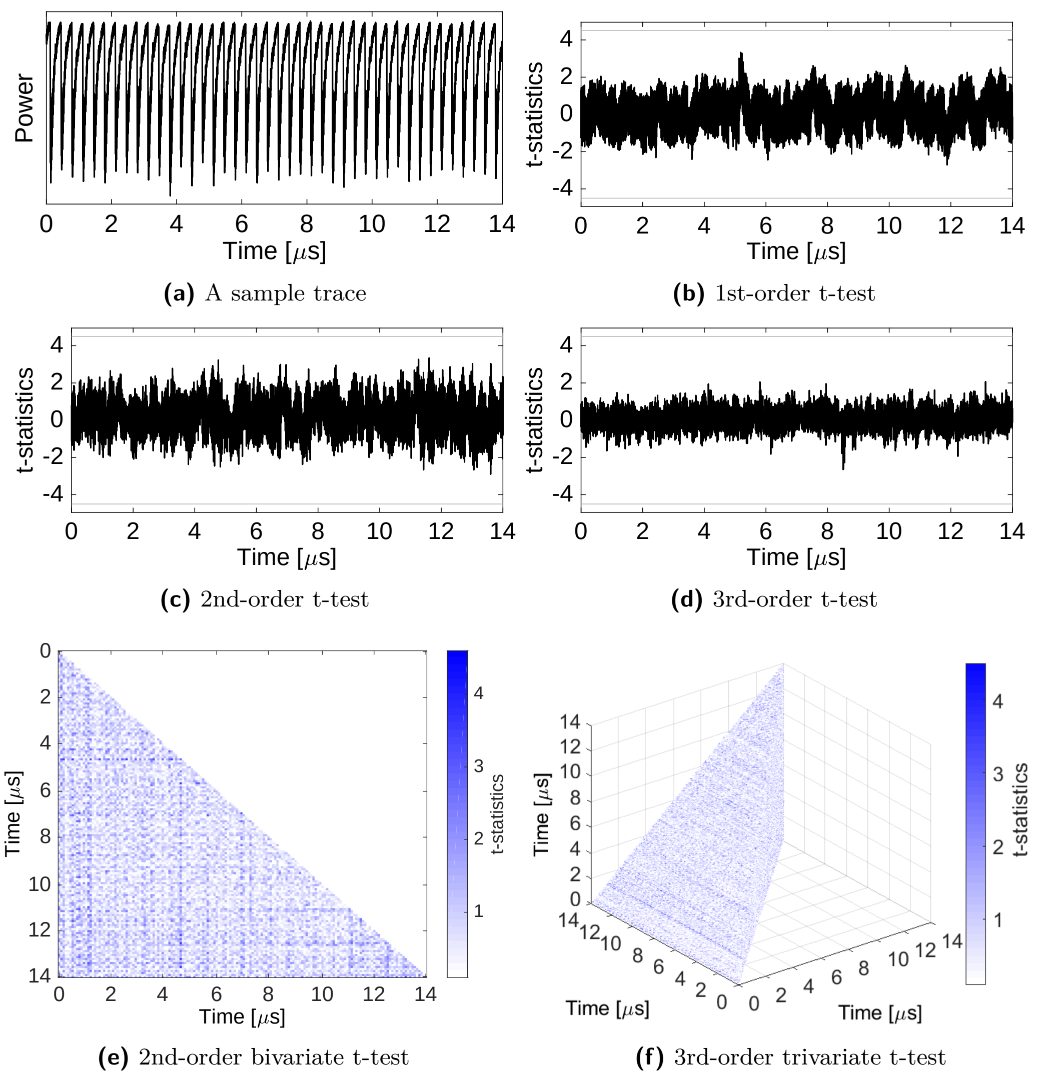
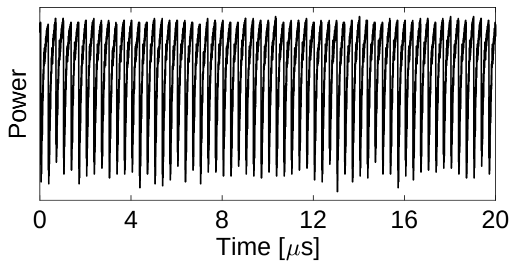
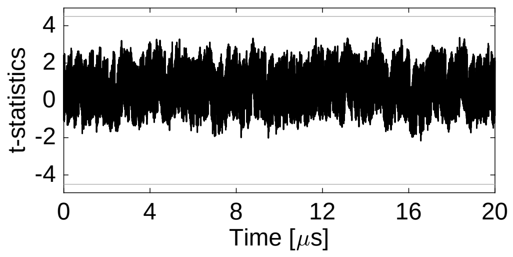


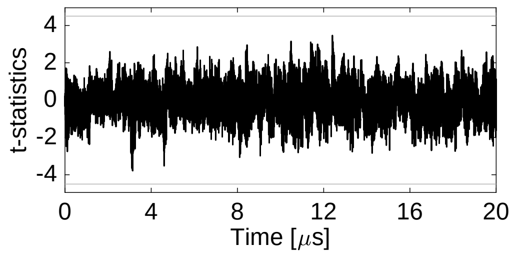
Figure 8: Experimental analysis of our masked MIDORI using 100 million traces.



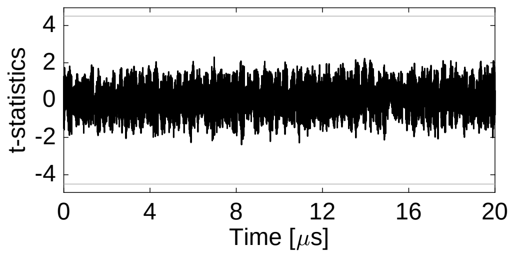
(a) A sample trace



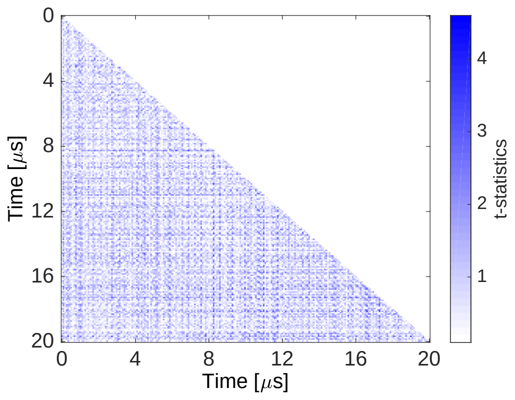
(b) 1st-order t-test



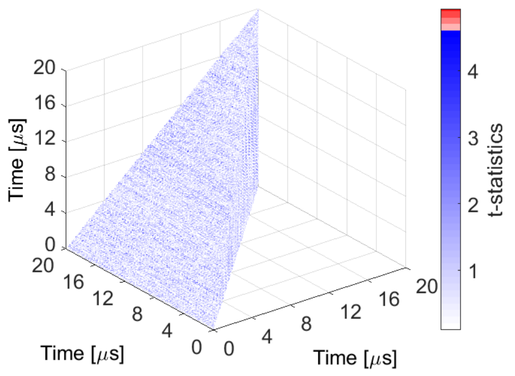
(c) 2nd-order t-test



(d) 3rd-order t-test



(e) 2nd-order bivariate t-test



(f) 3rd-order trivariate t-test

Figure 9: Experimental analysis of our masked SKINNY using 100 million traces.

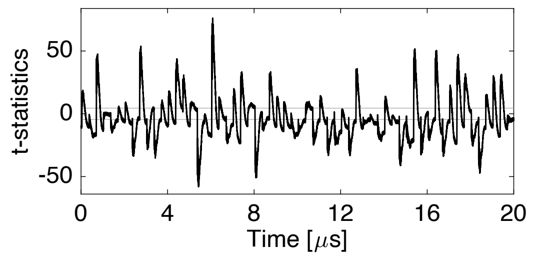


Figure 10: Experimental analysis (1st-order t-test) of our masked Skinny while both PRNG and initial sharing are off using 100 thousand traces, confirming the ability of our setup to detect 1st-order leakages.

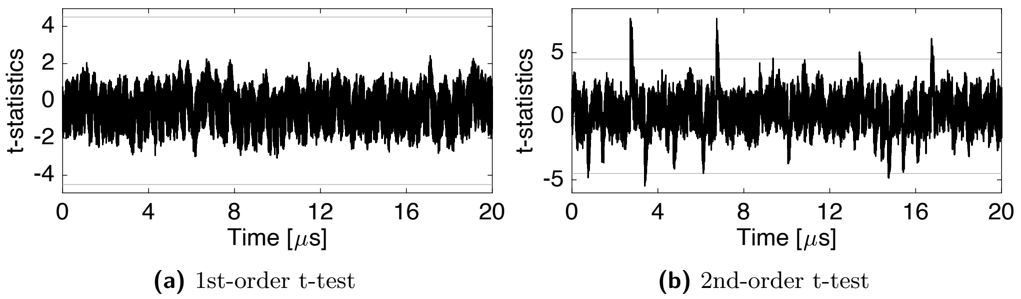


Figure 11: Experimental analysis of our masked Skinny while only PRNG is off using 100 million traces, confirming the ability of our setup to detect 2nd-order leakages