Generic Low-Latency Masking in Hardware

Hannes Gross, Rinat Iusupov, Roderick Bloem

Institute for Applied Information Processing and Communications (IAIK),
Graz University of Technology, Inffeldgasse 16a, 8010 Graz, Austria
hannes.gross@iaik.tugraz.at

Abstract.
In this work, we introduce a generalized concept for low-latency masking that is applicable to any implementation and protection order, and (in its most extreme form) does not require on-the-fly randomness. The main idea of our approach is to avoid collisions of shared variables in nonlinear circuit parts and to skip the share compression. We show the feasibility of our approach on a full implementation of a one-round unrolled Ascon variant and on an AES S-box case study. Additionally, we discuss possible trade-offs to make our approach interesting for practical implementations. As a result, we obtain a first-order masked AES S-box that is calculated in a single clock cycle with rather high implementation costs (60.7 kGE), and a two-cycle variant with much less implementation costs (6.7 kGE). The side-channel resistance of our Ascon S-box designs up to order three are then verified using the formal analysis tool of [BGI+18]. Furthermore, we introduce a taint checking based verification approach that works specifically for our low-latency approach and allows us to verify large circuits like our low-latency AES S-box design in reasonable time.

Keywords: masking · low latency · AES · hardware security · threshold implementations · domain-oriented masking

1 Introduction

Boolean masking is one of the most popular and well-studied countermeasures against side-channel analysis attacks, like differential power analysis [KJJ99] or electromagnetic emanation analysis [QS01]. The protection against these kinds of attacks, however, does not come for free. Masking hardware implementations requires additional circuitry which is partially spent on masking the actual implementation, but also on producing randomness to perform resharing and secure compression in nonlinear parts of the circuit. A lot of research over the last years has thus focused on reducing the hardware overhead either by making the masking itself more area efficient or by reducing the amount of required online randomness [BDN+14, BGN+14, BGN+15, GM17, GMK17, RBN+15]. Reducing the amount of randomness seems to go hand in hand with the need for a better control over the glitching behavior of circuits [BDF+17, GM17, GM18] which requires more registers and thus naturally introduces more latency.

In practice there exist many applications in which a short response time of the system as well as protection against side-channel analysis is indispensable. The research in the area of masking over the last years, however, has only marginally addressed low latency as a design goal. Most recently, Arribas *et al.* [ABP+17] introduced a first-order protected and two-round unrolled threshold implementation of Keccak, and Ghoshal *et al.* [GC17] introduced different variants of a first-order protected Boyar-Peralta AES S-box. The masked AES S-box requires between 3 and 4 cycles per S-box calculation making it the construction with the lowest cycle count in the literature. Other existing works either require more cycles or have higher randomness requirements [GM17, GMK17, CRB+16].

To the best of our knowledge, there exists no masked implementation of the AES S-box that requires less than three cycles up to now, and no generic scheme that covers protection for low-latency applications.

Our contribution. In this paper, we give answers to the intriguing question how randomness usage and chip area can be traded for less latency in hardware. For this purpose, we first introduce a generalized concept for low-latency masking that builds on the domain-oriented masking scheme [GMK17]. The approach is applicable to all security-sensitive circuits and is generic in terms of protection order. We then show the feasibility of our idea by applying the concept to a one-round unrolled masked implementation of the authenticated encryption scheme Ascon, and analyze the overhead over existing generically protected designs in terms of latency, chip area, and randomness consumption. Since Ascon was especially designed for efficient protection against side-channel attacks, we then target the AES S-box which has a higher algebraic degree and has proven to be a meaningful benchmark for the efficiency of masking algorithms.

We show three different designs of a masked AES S-box. The first S-box is purely combinatorial and thus cannot only be evaluated in one clock cycle, but even additional linear operations at the outputs are possible within the same clock cycle. The first-order masked S-box variant requires around 17.8 kGE and requires no additional online randomness (even for higher orders). The output shares, however, are increased from $d+1$ shares (for the S-box inputs) to $2(d+1)^7$. The second variant performs a share compression at the output which reduces the $2(d+1)^7$ output shares back to $d+1$ shares for the cost of 60.7 kGE and 2 kbits of fresh randomness. The third design again keeps the number of shares at $d+1$ by performing a share compression in the middle of the S-box and at the outputs. This variant has lower chip area (6.7 kGE) requirements but requires 416 bits of randomness per S-box calculation to keep $d+1$ shares. All designs can be generically scaled to the desired protection order. Finally, the security of our approach is first discussed and then analyzed using the formal analysis tool of [BGI+18] and a taint checking approach specifically designed for our low-latency approach.

In Section 2, we first give an introduction to Boolean masking and the domain-oriented masking scheme which we use as a basis for our approach. Our low-latency approach is then presented in Section 3 in which we show that masking not necessarily introduces latency or requires online randomness. In Section 4, we apply this approach to the Ascon S-box and discuss some potential pitfalls. We use this S-box construction to implement a round unrolled implementation of Ascon with generic protection and compare it to existing implementations. A single-cycle AES S-box is then introduced in Section 6 and trade-offs are discussed which lead to a two-cycle AES S-box with fewer hardware requirements. The analysis of the side-channel resistance of our implementations is performed in Section 8 before we draw conclusions and discuss future work in Section 9.

2 Introduction to Masking and Methodology

In general, masking works by disguising side-channel information of sensitive variables and intermediates by randomizing their representation. By randomizing the representation of variables, the side-channel information produced during the computation on these variables is made independent from the underlying data up to a certain degree. In the following, we use a sharing-based masking notation in which the information is assumed to be split into h number of randomly created and uniformly distributed shares representing *e.g.* the underlying variable x.

For a Boolean masking with $d+1$ fresh random shares, where d is the so-called protection order, we can represent a masked variable x as the sum over its shares x_i so that at all times $x = \sum_{i=0}^{d} x_i$ is fulfilled. The sharing is performed in such a way that only the combined information on all shares leak any information on x. All operations are then performed on the shares x_i in such a way that at no time any intermediate result is statistically dependent on more than one share of x to guarantee security in the so-called probing model of Ishai *et al.* [ISW03].

Probing model. The probing model introduced by Ishai, Sahai and Wagner is the de facto standard model in which the side-channel resistance of a Boolean masked circuit is analyzed. Informally speaking, a circuit is said to be d^{th}-order secure in the probing model, if an attacker with the ability to place up to d probing needles on to any wire or gate of a circuit (to continuously record the signal transitions over time) is not able to combine the recorded information to reveal any (unshared) critical information.

For example, the sharing of the variable x on its own, which consists of $d+1$ shares, is by definition d^{th}-order secure in the probing model because it would require more than d needles to collect all $d+1$ shares. The inherent goal of a masked circuit is to keep this independence throughout the entire circuit. While keeping this independence is trivial for linear operations, for which operations can be performed on all shares separately, nonlinear operations usually require more attention. To manage the rather complicated handling of nonlinear operations, different masking schemes have been introduced over the years that help in designing secure circuits. For the rest of the paper we consider and extend the ideas behind the so-called domain-oriented masking (DOM) scheme [GMK17].

Domain-Oriented Masking. The basic idea of domain-oriented masking is based on the assumption that any circuit which can be separated (shared) into $d+1$ completely independent circuits is trivially secure in the probing model if each of these circuits uses at most one share per security-critical variable. To achieve this separation, each circuit is therefore associated with a certain share index which is called the domain. We use the term domain and the associated subcircuit interchangeably. For example, the first domain is associated with the share index 0 and it uses only shares of security-critical variables with this index (*e.g.* x_0, y_0).

While this domain separation approach works for linear operations in a straightforward manner, nonlinear operations require communication across domain borders. For example, the nonlinear calculation of the finite field multiplication $x \cdot y$ in shared form (assuming $d+1$ shares per variable), requires to calculate $\sum_{i=0}^{d} \sum_{j=0}^{d} x_i \cdot y_j$ with respect to the probing model. In DOM, this is solved by using a DOM multiplier that follows the domain separation. A first-order variant ($d=1$) is shown in Figure 1.

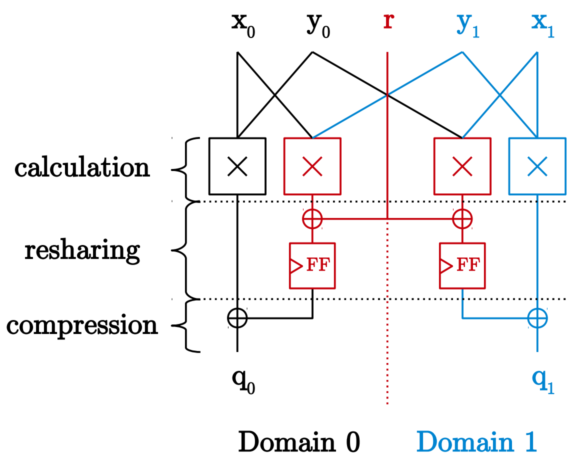

Figure 1: First-order DOM multiplier for independently shared inputs

According to the DOM scheme the circuit is split (vertically) into domain 0 and domain 1 which are colored black and blue, respectively. Horizontally, the circuit is split into three steps called calculation, resharing, and compression. During the calculation step, the so-called inner-domain multiplication terms, which use only shares with the same share index ($x_0 \cdot y_0$ and $x_1 \cdot y_1$), do not violate the domain separation and can thus securely be used in their respective domains. The red parts represent the so-called cross-domain terms ($x_0 \cdot y_1$ and $x_1 \cdot y_0$). These terms mix different domains and are potentially insecure due to what we refer to as variable collisions and look at more deeply in Section 3. The potential flaw can be easily observed if we assume the calculation of $x \cdot x$ with the same DOM multiplier circuit and the same sharings for both operands. We denote that the nonlinear combination of the same variable (a variable collision) is not necessarily intended by the designer of the circuit and can happen due to temporary logic states of gates, so-called glitches.

Both cross-domain terms of the DOM multiplier would then calculate $x_0 \cdot x_1$ which violates the probing model by combining all shares of x. If the dependency is only temporary, a register can be used in front of the multiplier to hinder the propagation of the glitches. Otherwise, if the dependency is permanent, a resharing step needs to be performed. For the remainder of this section, we assume that we have an independent sharing for the inputs of the multiplier. The resulting cross-domain terms are thus inherently secure in the probing model.

Due to efficiency reasons (to save registers and logic gates), a compression of the multiplication terms is usually performed also in other existing masked multiplication algorithms that operate on $d + 1$ input shares. This compression step is also performed in the depicted DOM multiplier by summing up the multiplication terms on the output. The result q is then again shared with only $d + 1$ shares. To allow the secure summing of the multiplication terms, a resharing step is performed by adding fresh randomness r to the two cross-domain terms. The subsequent use of a register ensures that the resharing is active before the terms are compressed on the output of the multiplier and suppresses the propagation of glitches. This resharing process allows to securely integrate the cross-domain terms without violating the domain separation requirement of DOM.

The introduced multiplier as well as the domain separation can be generically extended to any desired protection order. Independent from the protection order, the resharing step for the DOM multiplier always introduces one cycle of latency. For more details we refer the interested reader to [GMK17]. There also exist more randomness-efficient variants of the DOM multiplier that require up to five cycles for the secure compression [GM17, GM18]. In the next section, we extend the idea of domain separation and leverage it to reduce the latency in masked circuits.

3 Our Low-Latency Approach

As denoted in Section 2, the causes which hinder the calculation of a DOM masked circuit in fewer clock cycles are: 1) the compression to $d + 1$ shares after nonlinear operations (like the DOM multiplier in Figure 1), which require registers for the resharing of the cross-domain terms, and 2) the temporary or permanent dependencies (variable collisions) at the inputs of a nonlinear circuit parts. Our low-latency approach thus works by skipping share compression and avoiding variable collisions at the input of nonlinear functions.

Compression skipping. Our main observation is that the resharing and compression to $d + 1$ shares (and thus also the randomness and additional circuitry) is not a necessity induced by the probing model itself. It is solely performed for practical reasons and to some extent to make the result independent of the shared operands without having an explicit mask refreshing. We adapt the domain separation requirement of the DOM approach insofar as we still constrain each domain to use at most one share per variable but with the addition to allow domains with mixed share indices.

For example, the shared multiplication $q = x \cdot y$ in Figure 1 without a subsequent resharing and compression step thus results in four share domains for the result variable q (Figure 2). Each domain contains only one multiplication term from the calculation step. Any subsequent linear operations on the shares of q that only involve shares that are already used in the respective domain can be performed while maintaining security in the probing model. If q is multiplied by another variable, *e.g.* z with $d+1$ shares, the number of shares and domains grows to $(d+1)^3$ and so on. To keep this exponential blowup of shares and domains within reasonable bounds, the number of consecutive nonlinear operations needs to be minimized, or otherwise a secure share compression needs to be performed at some point, if the blowup becomes unacceptable.

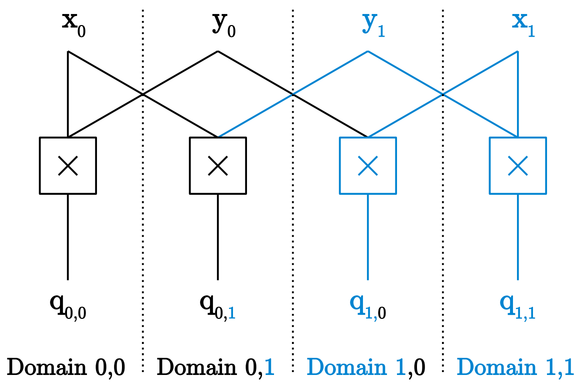

Figure 2: First-order low-latency multiplication with compression skipping, resulting in four domains

The security in the probing model for the compression skipping approach is given because any masked circuit that can be divided into at least $d+1$ independent subcircuits (without any wires to the other subcircuits), where each subcircuit uses at most one of the $d+1$ input shares from each variable, requires at least $d+1$ probes to combine all shares of one variable.

Avoiding variable collisions. The up to now tacit assumption that allows for the nonlinear combination of shares without compression and mask refreshing is that all operands have independent sharings. For example, it is relatively straightforward to apply this approach to a masked circuit that calculates $x \cdot y \cdot z$ if all involved shares are produced using independent and fresh randomness. The calculation of $(x \cdot y) \cdot x$, on the other hand, requires more attention (see Figure 3, left). One of the resulting multiplication terms would be $x_0 \cdot y_0 \cdot x_1$ ($q_{0,0,1}$) which brings two shares of x together and thus violates the domain separation requirement. This circumstance is indicated in Figure 3 by the different coloring of the shares for x which when combined result in an insecure sharing (colored red). One approach to circumvent the violation of the probing model that is used by the threshold implementations scheme, for instance, is to use more than $d+1$ shares and to ensure that in the worst case the probing attacker gets access to at most d shares when using up to d probing needles. Efficient sharings that fulfill the properties required by the TI [NRR06] scheme (correctness, independence, and uniformity) at the same time are, however, not trivial to find.

Instead of increasing the share count per variable, we propose the duplication of colliding variables (and gates) by using multiple shared instances of the same variable with independent sharings. Instead of calculating $(x \cdot y) \cdot x$, we thus calculate the equivalent

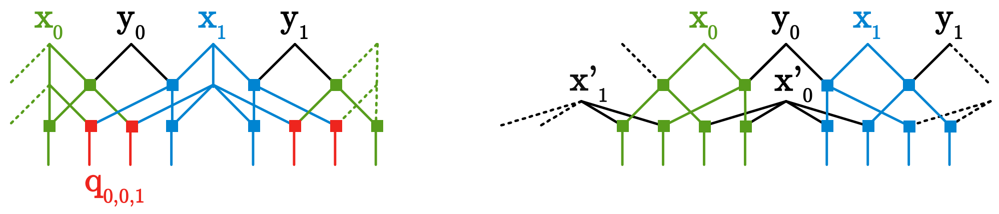

Figure 3: Example for an insecure first-order masked circuit, calculating $(x \cdot y) \cdot x$ (left), and a secure circuit $(x \cdot y) \cdot x'$ (right). The shares of x are colored green (x_0) and blue (x_1) for the sake of clarity.

$(x{\cdot}y){\cdot}x'$ where $x = \sum_{i=0}^{d} x_i = \sum_{i=0}^{d} x'_i$, and all involved shares are picked independently and uniformly at random. As Figure 3 (right) shows, mixing of the shares of x is circumvented this way. While at first sight this may seem as if we are using a sledgehammer to crack a nut, it has the same randomness costs for sharing a variable than *e.g.* a first-order ($d = 1$) TI with three shares and does not require additional (online) randomness. We would thus share x into $x_0 = x \oplus r_0$ and $x_1 = r_0$, and x' into $x'_0 = x \oplus r_1$ and $x'_1 = r_1$. The probing security for this simple example can be easily observed by writing down all resulting shares $q_{i,j,k} = x_i \cdot y_j \cdot x'_k$. Since none of the shares of q contain two shares of the same variable, the sharing is secure.

More generally, the probing security of any circuit with $d{+}1$ input shares and protection order d is given, if at no point in the circuit there exists a path from one share of a variable to another share of the same variable (assuming that all variables are independently shared). We will also use this condition for the taint checking based verification in Section 8 which allows us to perform a fast verification of the probing security of a circuit.

Resolving collisions caused by gates. When looking at complex circuits, the collisions can no longer entirely be resolved by duplication of the input variables. For the purpose of illustration, we consider a purely combinatorial unmasked circuit (Figure 4).

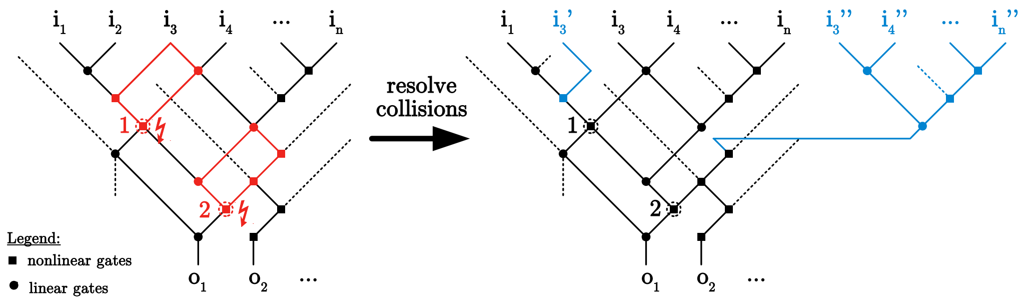

Figure 4: Example for collisions directly caused by inputs (1) and collisions caused by gates (2), collisions (left) and resolved collisions (right).

We note that collisions that would be caused when a circuit is masked using the DOM scheme are already evident in the unshared circuit. The first collision in this circuit (1) is caused because there exists a path (over other gates) from one input of the circuit to both inputs of a nonlinear gate. Since this collision is directly caused by the circuit input i_3, we could simply duplicate the input that causes the collision as before and connect the

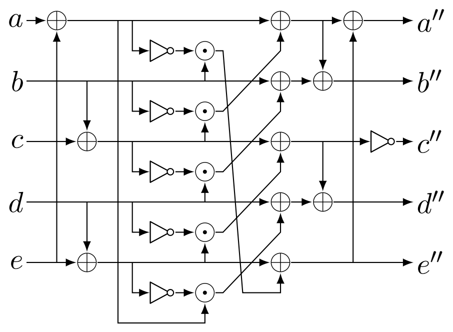

Figure 5: Ascon's original S-box, with collisions in a to d

copy (i'_3) accordingly (Figure 4, right). The second collision is caused by a gate (2) that has a path to both inputs of a nonlinear gate. In this case, simply duplicating the inputs would not be enough to avoid this collision. Instead, the gate causing the collision needs to be duplicated including its entire fan-in circuitry and its inputs. The output of the duplicated circuit then needs to be used in one path instead of the output wire of the gate that caused the collision.

In the next sections, we demonstrate the feasibility of our low-latency masking approach on practical examples and discuss trade-offs and possible pitfalls.

4 A Low-Latency Ascon S-box

As a first proof of concept, we introduce a masked Ascon S-box that can be evaluated in a single clock cycle while existing $d+1$ share implementations [GM17] require at least three clock cycles and up to seven cycles, respectively. The S-box is equivalent to the Keccak S-box except for an affine transformation on the input that produces temporary variable collisions making it a viable first practical example for our approach. We first transform the unshared S-box circuit to free the circuit from variable collisions, and then share the S-box according to our low-latency approach.

Collision-free S-box. The structure of the S-box is depicted in Figure 4, which corresponds to Equation 1.

$$\begin{aligned}
a'' &= (a \oplus e) \oplus \underline{(\neg b\,(b \oplus c))} \oplus (d \oplus e) \oplus (\neg\,(a \oplus e)\,b) \\
b'' &= b \oplus (\neg\,(b \oplus c)\,d) \oplus (a \oplus e) \oplus \underline{(\neg b\,(b \oplus c))} \\
c'' &= \neg\Big((b \oplus c) \oplus \underline{(\neg d\,(d \oplus e))}\Big) \\
d'' &= d \oplus \underline{(\neg\,(d \oplus e)\,(a \oplus e))} \oplus (b \oplus c) \oplus \underline{(\neg d\,(d \oplus e))} \\
e'' &= (d \oplus e) \oplus (\neg\,(a \oplus e)\,b)
\end{aligned} \tag{1}$$

By looking at the equations one can observe that there is a variable collision in the AND gates in five cases (underlined parts in Equation 1). These are the nonlinear gates that would produce a violation in the probing model due to glitches in case we shared the S-box (see Section 3). For example, $(\neg b\,(b \oplus c))$ in a'' combines the variable b with itself in an AND gate which would combine shares with different share index (*e.g.* b_0 and b_1) when the S-box is shared and thus create a violation.

To avoid collisions in the AND gates, we duplicate the signals b, d, and e (b', d', and e') and replace one of the operands of the AND gates accordingly as shown in Equation 2.

$$\begin{aligned}
a'' &= (a \oplus e) \oplus (\neg b' \, (b \oplus c)) \oplus (d \oplus e) \oplus (\neg \, (a \oplus e) \, b) \\
b'' &= b \oplus (\neg \, (b \oplus c) \, d) \oplus (a \oplus e) \oplus (\neg b' \, (b \oplus c)) \\
c'' &= \neg \, ((b \oplus c) \oplus (\neg d' \, (d \oplus e))) \\
d'' &= d \oplus (\neg \, (d \oplus e) \, (a \oplus e')) \oplus (b \oplus c) \oplus (\neg d' \, (d \oplus e)) \\
e'' &= (d \oplus e) \oplus (\neg \, (a \oplus e) \, b)
\end{aligned} \tag{2}$$

Sharing of the S-box. Since the S-box description is now free from any variable collisions, the S-box can safely be shared following our low-latency approach. We assume that each of the five inputs and the two duplicated inputs are shared using $d+1$ shares. As there is a single layer of AND gates, the shares and thus the domains for each of the outputs grow from $d+1$ to $(d+1)^2$, and we use two indices (i and j) to denote the according output share.

$$\begin{aligned}
a''_{i,j} &= \begin{cases} (a_i \oplus e_i) \oplus \left(\neg^i b'_i \, (b_i \oplus c_i)\right) \oplus (d_i \oplus e_i) \oplus \left(\neg^i \, (a_i \oplus e_i) \, b_i\right), & \text{if } i = j. \\ \left(\neg^i b'_i \, (b_j \oplus c_j)\right) \oplus \left(\neg^i \, (a_i \oplus e_i) \, b_j\right), & \text{otherwise.} \end{cases} \\
b''_{i,j} &= \begin{cases} b_i \oplus \left(\neg^i \, (b_i \oplus c_i) \, d_i\right) \oplus (a_i \oplus e_i) \oplus \left(\neg^i b'_i \, (b_i \oplus c_i)\right), & \text{if } i = j. \\ \left(\neg^i \, (b_i \oplus c_i) \, d_j\right) \oplus \left(\neg^j b'_j \, (b_i \oplus c_i)\right), & \text{otherwise.} \end{cases} \\
c''_{i,j} &= \begin{cases} \neg^i \left((b_i \oplus c_i) \oplus \left(\neg^i d'_i \, (d_i \oplus e_i)\right)\right), & \text{if } i = j. \\ \left(\neg^i d'_i \, (d_j \oplus e_j)\right), & \text{otherwise.} \end{cases} \\
d''_{i,j} &= \begin{cases} d_i \oplus \left(\neg^i \, (d_i \oplus e_i) \, (a_i \oplus e'_i)\right) \oplus (b_i \oplus c_i) \oplus \left(\neg^i d'_i \, (d_i \oplus e_i)\right), & \text{if } i = j. \\ \left(\neg^i \, (d_i \oplus e_i) \left(a_j \oplus e'_j\right)\right) \oplus \left(\neg^j d'_j \, (d_i \oplus e_i)\right), & \text{otherwise.} \end{cases} \\
e''_{i,j} &= \begin{cases} (d_i \oplus e_i) \oplus \left(\neg^i \, (a_i \oplus e_i) \, b_i\right), & \text{if } i = j. \\ \left(\neg^i \, (a_i \oplus e_i) \, b_j\right), & \text{otherwise.} \end{cases}
\end{aligned} \tag{3}$$

For each output variable we consider two cases:

1) The case $i = j$ covers the inner-domain terms where only variables with the same share index appear. To ensure correctness of the sharing, the negation *e.g.* $\neg^i$ is only effective if the corresponding variable in the superscript equals zero, so that only the first share of a variable is inverted.

2) For the remaining output shares, we need to be more careful to fulfill the domain separation requirement. By the duplication of the according inputs we ensured that there are no two paths for any of the input variables that are combined in a nonlinear AND gate, which would result in a flaw that could not be avoided in this case. However, for linear gates we still need to ensure that we do not combine shares with different share indices from the same variable in the same domain (domain separation requirement). For example, $(b'_i \, (b_j \oplus c_j)) \oplus ((a_i \oplus e_i) \, b_j)$ in a'' would produce a flaw in case we switched the share index variable of one of the b variables (i to j) in this equation so that we have $(\ldots (b_i \ldots)) \oplus ((\ldots b_j)$. For this reason, we also need to set the indices in b'' and d'' for the last AND gate terms accordingly.

The correctness of the sharing is given by the fact that the sums over i and j over each output variable result in Equation 1 when b' is set to b, d' is set to d, and e' is set to e. The security is given by the fact that we do not have any domain crossings (as verified in Section 8).

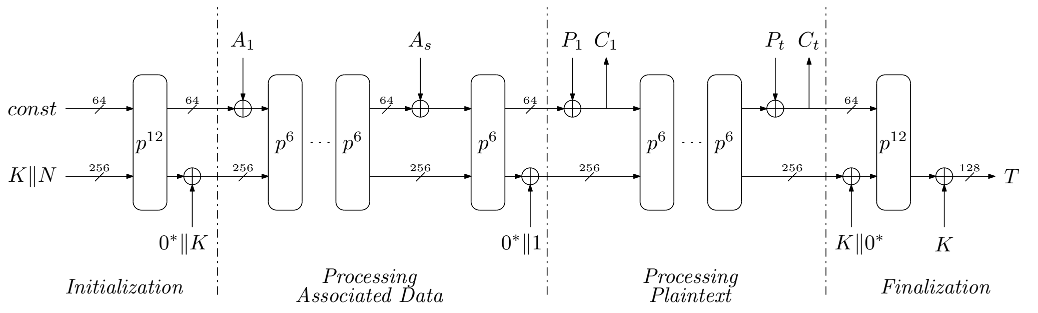

Figure 6: Data encryption and authentication with ASCON

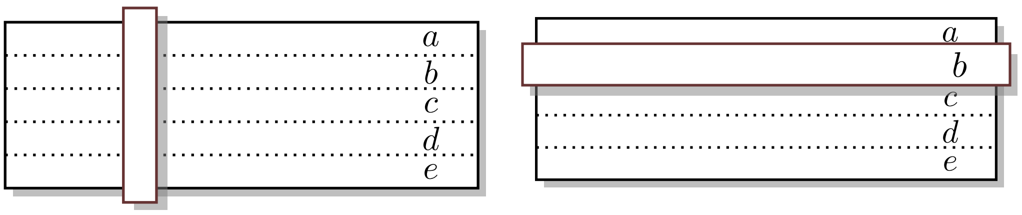

Figure 7: Column-wise application of ASCON's S-box to the state (left) and row-wise for the linear layer (right), duplicated state rows are omitted

5 A Low-Latency Variant of Ascon

In this section, we integrate the low-latency S-box design into a round-unrolled variant of ASCON-128. The sponge mode for the data encryption and authentication is depicted in Figure 6, in which the round transformation p is performed iteratively either twelve (p^{12}) or six (p^6) times on the state in each round. One round transformation consists of three parts $p = p_L \circ p_S \circ p_C$. The linear round constant addition p_C followed by the S-box layer p_S (Figure 7, left) and the linear transformation layer p_L (Figure 7, right). Since the S-box layer in ASCON is only preceded by the linear addition of key or data, and only followed by the linear transformation layer (which both can be securely realized by only operating on each share separately), the shared S-box description from the last section can now be used to implement a full round transformation of ASCON without any registers in between.

For the sake of completeness, we remark that the combination of the shares created by the shared S-box in Equation 3, *e.g.* of a'' and b'', would not be secure because different share indices are used for some variables (the term $b_j \oplus c_j$ in a'' collides with the term $b_i \oplus c_i$ in b''). However, this is not an issue for the one-round unrolled ASCON variant because the S-box is calculated column-wise over the state (as shown in Figure 7, left) and is only followed by a linear transformation that operates inside one state row (right). Independence of the cipher rounds is ensured by a resharing steps after each round transformation. The resharing step also includes the creation of the duplicated state rows for the next round by applying the compression two times in parallel for these variables with fresh and independent randomness from the random number generator (RNG).

Design description. Figure 8 depicts our top module of the ASCON core. The structure is based on the one used by Gross *et al.* [GM17] for which the sources are available online [Gro18]. The majority of changes are done in the state module (right). The round transformation is no longer distributed over (at least) three clock cycles but is performed in a single step. Due to the S-box layer, the amount of shares increases from $d+1$ to $(d+1)^2$ for the linear layer which is followed by a remasking step according to the CMS scheme of

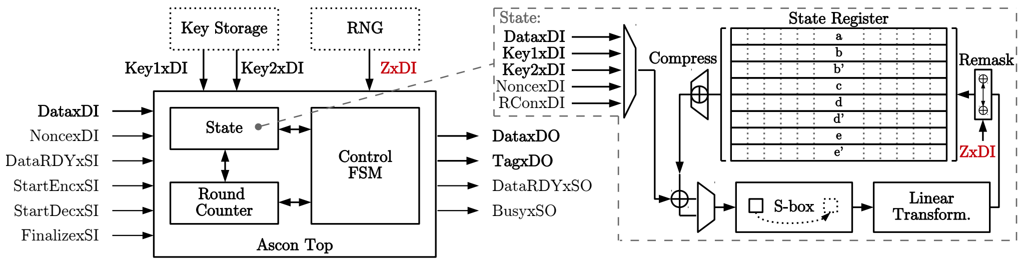

Figure 8: Hardware design overview of Ascon

Reparaz *et al.* [RBN+15]. The CMS remasking requires one fresh random bit per share which amounts to $8 \cdot 64 \cdot (d+1)^2$ bits in total for our design. Before the compression to $d+1$ terms can be performed, the $(d+1)^2$ refreshed shares are stored in the state registers which include duplicated state rows needed for the S-box layer in the next round transformation. The number of state registers is therefore increased from $5 \cdot 64 \cdot (d+1)$ to $8 \cdot 64 \cdot (d+1)^2$ compared to [GM17] which is partially compensated by the registers which are not required for the S-box layer.

Another change affects the key storage which now needs to supply an additional copy of the key since the key is combined with the state during the initialization and the finalization (see Figure 6) and is used in parts of the state that need to be copied for the secure S-box transformation.

Results and comparison. The post-synthesis results for a 90 nm Low-K UMC process with 1 V supply and a 20 MHz clock synthesized with the Cadence Encounter RTL compiler v14.20 of the low-latency designs are given in Table 1. The design is generic in terms of protection order (d), but since the number of registers grows quadratically with the protection order, we only considered results up to order five. For all protection orders, only six cycles per encryption or decryption are required which is three to seven times fewer than for the DOM and UMA designs (with 64 parallel S-boxes) in [GM17]. We note that the reason why the DOM and UMA designs require more clock cycles for the calculation of one round is not due to an optional design decision, e.g. to shorten the longest combinatorial path, but is first of all a security necessity of the used masked AND gates. The linear input transformation at the beginning of the S-box creates multiple variable collisions that would lead to a flaw in the probing model due to glitches in case no register was used after the input transformation (see Figure 4). This results from the fact that the used masked AND gates are only secure if both input operands are independently shared which is not the case due to these (temporary) variable collisions. We note the existence of DOM-dep. masked AND gates (see [GMK17]) which are resilient to variable collisions for the price of more randomness and chip area. These could be used to avoid this register stage thus reducing the cycle count by one. However, in either case the DOM masked AND gates require at least one register stage for the resharing step, and the UMA masked AND gate requires one to five register stages. Linear transformations within the resharing stage other than the actual resharing with a fresh random mask are not considered neither in the original DOM scheme nor in UMA.

Unrolling an entire round of course produces much more combinatorial delay which results in a lowered maximum clock frequency. The maximum clock frequency for our first-order implementation, for example, is only about 260 MHz while the DOM and UMA variants achieve a clock frequency of more than 630 MHz. When considering the latency as the time from one plain text block being fed into the Ascon-128 core until the cipher text is available on the output (see Table 1), it shows that our approach saves with 23.1 ns latency for one encryption almost 20% over the first-order DOM and UMA implementations with 28.44 ns at maximum clock speed. For the fifth protection order, the differences grow to about 38% and 72%, respectively. Also the maximum throughput is in all cases increased over related work. While again for first-order the throughput is only slightly increased, the difference becomes much more significant for order five for which the throughput is almost doubled over the DOM design and 3.5 times higher than for the UMA design. The price for the reduced latency is an increased chip area (about 15 kGE overhead for the first-order variant, and double the amount of area over DOM for order five), and an increased randomness consumption which is between 5.2 (UMA, order five) and 6.4 (DOM/UMA, first order) higher.

Table 1: Results for Ascon-128 with one cycle per round (64 S-boxes)

Design	Size *[kGE]*	Cycles */Round*	Max. Throughput *[Gb/s]*	Randomness *[bits/cycle]*	Latency *[ns]*
Unprotected	8.35	1	5.84	-	10.96
1st-order	42.75	1	2.77	2,048	23.10
2nd-order	90.94	1	3.35	4,608	19.10
3rd-order	153.91	1	3.34	8,192	19.16
4th-order	238.30	1	2.59	12,800	24.71
5th-order	339.82	1	2.99	18,432	21.40
Related work [GM17]					
1st-order UMA	27.18	3	2.25	320	28.44
1st-order DOM	28.89	3	2.25	320	28.44
5th-order DOM	161.87	3	1.86	4,800	34.14
5th-order UMA	220.01	7	0.85	3,520	75.29

Discussion. We admit that the randomness requirements for the higher-order variants become very high but we denote two things:

1) Our low-latency approach offers a new design choice that a designer of a masked circuit can use to trade off area and randomness against less latency. We used one corner case to demonstrate the feasibility of the approach by targeting one cycle per round transformation. A designer, of course, could also target a two-cycle variant by using the resharing *e.g.* after the S-box or by inserting registers after the affine transformation in the S-box to save randomness and area. We discuss trade-offs in more detail for the AES S-box in the next section.

2) The CMS resharing function is probably not an ideal choice. A DOM-like resharing, for example, could possibly reduce the randomness amount, *e.g.* for first order by a factor of 4 which would reduce the randomness to 512 bits per cycle. On the other hand, using the DOM resharing would require a deeper analysis of the design over at least two rounds, while the CMS resharing separates the rounds by resharing all bits of the state before the next round starts. Therefore, we made the choice to use the CMS resharing at this point and denote the use of a more efficient resharing function as one interesting practical extension of our work.

6 A Low-Latency Masked AES S-box

The efficient (masked) implementation of the AES S-box has proven to be a difficult practical problem and a huge variety of papers have been published on S-box constructions.

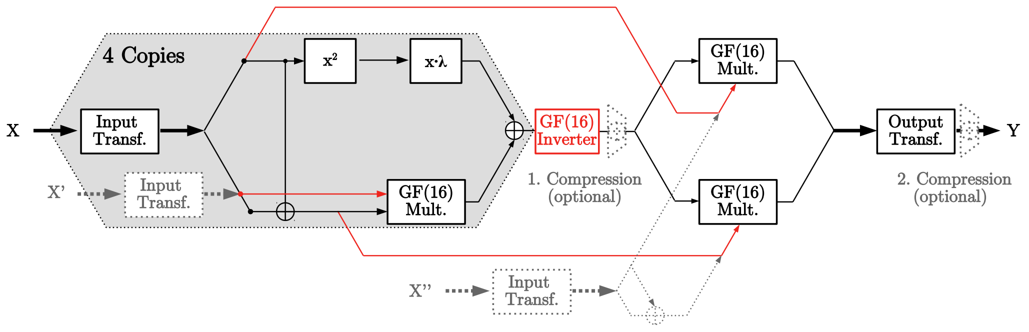

Figure 9: Mui S-box design (black and red parts are from the original design), gray dotted paths and elements replace the red paths to which they are connected in the collision-free design

Most of the recent work on masked AES implementations takes the S-box design of Canright [Can05] as a basis. The original design goal of Canright's S-box design is low chip area for an unmasked implementation which does not automatically result in the lowest area costs for a side-channel protected implementation. For our low-latency approach, the maximum logic depth and in particular the nonlinear gate depth (number of AND gates or GF multipliers in the logic path) seems to be the natural major design criterion because at each nonlinear gate the number of shares is increased. The S-box design of Boyar and Peralta [BP12] addresses low logic depth which results in a total logic depth of 16 and a nonlinear gate depth of 4. This design was most recently used in another work on low-latency masking by Ghoshal *et al.* [GC17] and has a latency of three to four cycles. Canright's S-box on the other hand has a logic depth of 25 to 27, and a nonlinear gate depth of 4 (in the variant as it is used by most masked implementations). Another important aspect that needs to be taken into account for our approach is the number of bit collisions because it determines the number of input duplicates we need to provide in order to guarantee collision freeness.

Choosing the most promising S-box design. As analyzing a circuit with respect to its collision behavior is rather time-consuming, we developed a tool that simply traces all inputs and gate outputs through a given circuit and checks for conflicts. We analyzed the Canright S-box (original design), the Boyar-Peralta S-box and the design of Edwin NC Mui [MCE07]. As it turns out, even despite the fact that the Boyar-Peralta S-box was designed for low circuit depth, it involves lots of gate dependencies which require quite a number of sub-circuit copies and input copies. Furthermore, the Canright and the Mui S-box designs do not break down the complete design of the AES S-box into single gates but rather consist of larger self-contained structures like Galois field multipliers which can be shared more efficiently than by sharing each AND gate separately. The circuit that showed the fewest dependencies is the design presented by Mui which we then chose to take as a basis for our masked design. However, we note that we do not claim this choice of the S-box or our low-latency implementations of it to be optimal.

6.1 "Zero Latency" AES S-box

Mui's design is depicted in Figure 9. The black and red (security-critical) paths correspond to the original design by Edwin NC Mui. The gray dotted circuit elements are used for the collision-free S-box design and replace the red paths. For the design of the S-box without collisions, we took an iterative approach for which we implemented the circuit from the inputs onwards to the next nonlinear part of the circuit and checked for collisions. We thus also split the explanation accordingly.

S-box inputs to inverter. After the input transformation that maps the S-box input x, which is interpreted as a polynomial in $GF(256)$, to two elements in $GF(16)$ the transformed input is split into two halves. The two halves are nonlinearly combined in the $GF(16)$ multiplier. Since the linear input mapping and the XOR in front of the first GF multiplier mix many of the input bits (cf. [MCE07] for details), it requires to duplicate all bits of x (x') except for the fifth bit and the circuitry that causes the flaw (the gray and dotted input mapping). Otherwise, an input collision would be caused in the multiplier as indicated by the red wire. For the shared S-box variant, the number of shares is increased from $d+1$ to $(d+1)^2$ after the multiplier and the linearly transformed parts (x^2 and $x\lambda$) are added with respect to their share domain.

GF(16) inverter. In Mui's S-box design, the $GF(16)$ inverter is given as Boolean equation instead of finite field arithmetic as *e.g.* in Canright's S-box. The mathematical description is stated in Equation 4. The inversion in $GF(16)$ results in collisions for all S-box input bits which requires to separate the calculation of all input bits of the inversion by copying the fan-in circuit (dotted gray hexagon, "4 Copies") four times including the changes as described above. Up to this point, the S-box circuit requires in total four full copies of the input x and four partial copies (x', each bit except for the fifth bit) to avoid collisions.

$$\begin{aligned}
a' &= a \oplus abc \oplus ad \oplus b \\
b' &= abc \oplus abd \oplus ad \oplus b \oplus bc \\
c' &= a \oplus abc \oplus acd \oplus b \oplus bd \oplus c \\
d' &= \underline{abc} \oplus \underline{abd} \oplus ac \oplus \underline{acd} \oplus ad \oplus b \oplus bc \oplus bcd \oplus c \oplus d
\end{aligned} \tag{4}$$

In contrast to the ASCON S-box example, the equations for the inverter are free from any internal collisions of the inverter inputs (there is no path from one input variable to both inputs of an AND gate). In order to avoid the combination of two or more shares of one input for the shared S-box representation, care needs to be taken also for the linear gates. Again we avoid collisions in the linear parts by associating with each variable one share index which we keep throughout the entire calculation. To keep the number of output shares to a minimum, we try to use as few share indices as possible. However, as can already be observed in the underlined parts of the unshared calculation of d', using only three indices is not always possible.

Reduced example for flawed indexing. To demonstrate the resulting problem for d' in the shared variant, we consider a reduced example that contains only the problematic parts:

$$q = abc \oplus abd \oplus acd$$

If we want to calculate the shared representation of q, we need to combine all shares (given by the indices i, j, and k) of the variables connected by an AND gate as given in the following example. We assume, as for the inverter inputs, that the input share count is already increased to $(d+1)^2$.

$$q_{(i,j,k)} = a_i b_j c_k \oplus a_i b_j d_k \oplus a_i c_j d_k$$

The problem arises in the XOR gates because we combine shares from the same variable c one time with the share index k and another time with index j which violates the mixed domains assumption. Since there is no way to overcome this issue by associating the share indices differently, the calculation is split into two parts. Splitting up the calculation in two parts as shown in Equation 5 increases the amount of shares from $(d+1)^2$ to $2(d+1)^6$ (the curly braces indicate a concatenation of shares).

$$q_{(i,j,k)} = \{a_i b_j c_k \oplus a_i b_j d_k, a_i c_j d_k\} \tag{5}$$

By applying this solution to the equation of the inversion (Equation 4), we can denote the sharing of the inverter as in Equation 6. The curly braces under the equations ensure correctness of the sharing and denote that certain terms are only present in certain output shares (when the stated constraint is fulfilled).

$$\begin{aligned}
a'_{(i,j,k)} &= \underbrace{a_{(i)}}_{j=k=0} \oplus a_{(i)}b_{(j)}c_{(k)} \oplus \underbrace{a_{(i)}d_{(k)}}_{j=0} \oplus \underbrace{b_{(j)}}_{i=k=0} \\
b'_{(i,j,k)} &= a_{(i)}b_{(j)}c_{(k)} \oplus a_{(i)}b_{(j)}d_{(k)} \oplus \underbrace{a_{(i)}d_{(k)}}_{j=0} \oplus \underbrace{b_{(j)}}_{i=k=0} \oplus \underbrace{b_{(j)}c_{(k)}}_{i=0} \\
c'_{(i,j,k)} &= \{\underbrace{a_{(i)}}_{j=k=0} \oplus a_{(i)}b_{(j)}c_{(k)} \oplus \underbrace{b_{(j)}}_{i=k=0} \oplus \underbrace{c_{(k)}}_{i=j=0}, a_{(i)}c_{(j)}d_{(k)} \oplus \underbrace{b_{(i)}d_{(k)}}_{j=0}\} \\
d'_{(i,j,k)} &= \{a_{(i)}b_{(j)}c_{(k)} \oplus a_{(i)}b_{(j)}d_{(k)} \oplus \underbrace{a_{(i)}c_{(k)}}_{j=0} \oplus \underbrace{a_{(i)}d_{(k)}}_{j=0} \oplus \underbrace{b_{(j)}}_{i=k=0} \oplus \underbrace{b_{(j)}c_{(k)}}_{i=0} \oplus \underbrace{c_{(k)}}_{i=j=0} \\
&\quad \oplus \underbrace{d_{(k)}}_{i=j=0}, a_{(i)}c_{(j)}d_{(k)} \oplus b_{(i)}c_{(j)}d_{(k)}\}
\end{aligned} \tag{6}$$

Final multiplier stage to output transformation. For the final multiplier stage we avoid collisions by using an additional set of freshly masked copies of the S-box inputs (x'', with $d+1$ shares). These copies are then combined with outputs of the $GF(16)$ inverter in the multipliers. As these multiplications happen in parallel and no nonlinear transformation follows in the S-box, only one additional copy of the inputs x'' suffices for both multiplications. The adjacent linear transformations are applied share-wise and with respect to the share domains to avoid collisions at this stage. Up to this point, no additional online randomness or registers are required. However, the number of shares is increased to $2(d+1)^7$ at this point. We call this variant the "zero latency" variant which denotes that further linear operations on the output shares (like *ShiftRows*, *MixColumns*, or *AddRoundkey*) are still possible within the same cycle. A share compression is thus not taken into account for this variant at this point. The first-order zero latency S-box requires 17.83 kGE of chip area for a 90 nm UMC process with a maximum clock frequency of 228 MHz.

6.2 Single-Cycle Variant

For the one cycle variant of the S-box we compress the shares at the output of the zero latency S-box by using a CMS compression function after the output transformation. The number of shares is thus reduced from $2(d+1)^7$ to $d+1$ again which requires $16(d+1)^7$ random bits and registers in total. This variant would help in a full implementation of the AES to reduce the number of subsequent linear transformations and registers, for the cost of the CMS resharing stage. The chip area requirements for the single-cycle masked S-box variant with first-order protection are 61 kGE at a maximum clock frequency of 356 MHz, and it requires 2 kbits of fresh randomness.

6.3 Two-cycle variant

The chip area costs for the single-cycle S-box variant are admittedly very high given the fact that one round unrolled variant of AES-128 requires 16 of these S-boxes. The costs can be reduced by performing an intermediate resharing and compression step after the inverter in Figure 9. The number of shares is thus reduced from $2(d+1)^6$ to $d+1$ before the last two multiplications are performed which saves many of the area-consuming $GF(16)$ multipliers and linear transformations at the output. The final compression requires $8(d+1)^2$ fresh random bits. In total, this variant requires $6(d+1)^6 + 8(d+1)^2$ random bits (416 bits for first-order protection) and the chip area is reduced to 6.7 kGE. For second order, the amount of required randomness is 4,446 bits and the chip area is 57 kGE.

7 Summary of the AES S-box Results and Comparison

A summary of the results for our low-latency AES S-box variants and related work is given in Table 2. All of our stated results are post-synthesis results for a 90 nm Low-K UMC process with 1 V supply and a 20 MHz clock, synthesized with the Cadence Encounter RTL compiler v14.20. The used cell library and tool chain vary among the stated related work and the numbers should not be compared directly.

As the comparison shows, our low-latency AES S-box variants are the first published constructions that reduce the latency below three cycles per S-box calculation. The price is a significant increase of both chip area and randomness requirements, especially for the single-cycle S-box variant with 60.73 kGE and 2 kbit of randomness. The zero latency variant requires with 17.8 kGE almost nine times more area than the smallest design. The chip area overhead for the first-order AES S-box with two cycles is relatively moderate with about a factor of three times the area of the smallest known S-box construction. Furthermore, our designs are generic like the DOM [GMK17] AES variant.

Comparing the randomness requirements is difficult since most of the stated work uses a different amount of input shares which is usually not considered to be part of the required (online) randomness. In this context, our zero latency variant requires no additional online randomness but it requires of course additional randomness for the sharing and the duplication of the input variables. In case of our two-cycle variant, the online randomness costs for calculating one S-box are with 416 (and 4,446 bits, respectively) significantly increased over the state of the art.

However, we note that our primary goal was to demonstrate that for generic higher-order protection a reduction of the latency is indeed possible even in complex designs like the AES S-box. The most efficient desigissuen choices and the best point at which the shares can be again compressed remains to be an open problem.

Discussion on the impact on full AES implementations. In the following, we want to briefly discuss the expected impact on the latency and throughput of a full AES-128 implementation on the basis of our S-box implementation results. We denote that we are fully aware of the fact that the provisioning of such high amounts of randomness, as required for multiple instances of our S-box implementations in parallel, as well as the required chip area and power consumption would exceed the capabilities of most practical applications. This comparison should merely serve as a basis for future comparisons and to demonstrate that a cycle count reduction not automatically leads to a reduction of the overall latency.

The impact on the throughput as well as on the latency highly depends on the concrete AES implementation whose assumptions can highly vary. For this reason, we make runtime estimations for two different corner cases, namely best-case and worst-case cycle count estimations. An AES-128 encryption consists of a pre-round (which only performs the

Table 2: Results and comparison of masked AES S-box implementations

Design	Order *[d]*	Size *[kGE]*	Cycles */S-box*	Max. Clock *[MHz]*	Randomness *[bits] (online)*
Zero Latency	first	17.83	0	228	0
Zero Latency	d		0		0
Single Cycle	first	60.73	1	356	2,048
Single Cycle	d		1		$16(d+1)^7$
Two Cycle	first	6.74	2	584	416
Two Cycle	second	57.11	2	517	4,446
Two Cycle	d		2		$6(d+1)^6+8(d+1)^2$
			Related work		
[BGN+14]	first	3.71	3		44
[BGN+15]	first	2.84	3		32
[CBR+15]	second	7.9 - 11.2	6		126
[CRB+16]	first	1.98	6		54
[GC17]	first	4.61	4		0
[GC17]	first	3.63 - 3.80	4		34 - 68
[GC17]	first	2.91 - 3.34	3		20-24
[GMK17]	first	2.2	8		18
[GMK17]	second	4.5	8		54
[GMK17]	d		8		9d(d+1)
[MPL+11]	first	4.24	4		48

AddRoundKey transformation) followed by nine full rounds, and the final round (which omits MixColumns). A full round consists of SubBytes (the S-box layer) followed by ShiftRows, MixColumns and AddRoundKey.

An overview on our cycle count estimation is given in Table 3. For the best-case runtime estimation, we assume that only the S-box layer introduces delay cycles due to non-linear calculations and that a full SubBytes transformation with 16 S-boxes in parallel is implemented. All other transformations are assumed to be performed implicitly and the round keys are already precomputed. For the best case, we thus estimate a runtime that is ten times (number of full rounds plus the final round) the delay of the used S-box transformation ($10l_{sbox}$). This cycle count corresponds, for example, to Intel's AES instructions [Gue09].

Table 3: Cycle count estimation for full AES-128 hardware implementations with a variable numbers of cycles for the S-box (l_{sbox})

Round	SubBytes	ShiftRows	MixColumns	AddRoundKey	Key Schedule
0	–	–	–	$0 \dots 16$	$0 \dots 2 + 16l_{sbox}$
$1 \dots 9$	$(1 \dots 16)l_{sbox}$	$0 \dots 1$	$0 \dots 4$	$0 \dots 16$	$0 \dots 2 + 16l_{sbox}$
10	$(1 \dots 16)l_{sbox}$	$0 \dots 1$	–	$0 \dots 16$	–
Overall	$10l_{sbox} \dots 224 + 320l_{sbox}$				

For the worst-case cycle count estimation, we assume that only one S-box is implemented in hardware, that the key schedule is performed on the fly, and that all other transformations require one clock cycle. The key schedule is thus assumed to require at most two cycles (for RotWord and Rcon) plus 16 times the number of the assumed S-box delay for SubWord. Our worst-case estimation of $224 + 320l_{sbox}$ approximately corresponds to Feldhofer et al.'s [FWR05] low-power AES implementation which requires 1,032 cycles for a two-cycle S-box.

For our single-cycle S-box variant, we thus estimate the cycle count for a full AES-128 to be between 10 cycles and 544 cycles, and for the two-cycle S-box the estimations result in 20 cycles and 864 cycles, respectively. Given the maximum clock frequencies in Table 2 (and neglecting additional combinatorial delay introduced by other components of the AES), the estimated latency for the single-cycle S-box AES is between 1.53 µs and 28.09 ns, and for the two-cycle S-box AES variant between 1.48 µs and 34.25 ns. The reduction to a single-cycle S-box could thus, in the best of cases save about 18% of latency while for the worst case estimation the single-cycle S-box introduces an even 3.3% higher latency in the full AES than the two cycle S-box. Similarly, the estimated throughput is between 83.76 Mbps and 4.56 Gbps for the single-cycle variant, and between 86.52 Mbps and 3.74 Gbps for the two-cycle S-box variant.

In summary, even when keeping practical limitations regarding randomness provisioning, chip area, and power or energy consumption aside, it is not entirely clear whether or not a reduction of the latency for the nonlinear parts of a cryptographic implementation automatically leads to a reduction of the overall latency of the system. In practice, the most valuable design choices therefore need careful evaluation of the overall constraints of a system.

8 Analyzing the Side-Channel Resistance

For the analysis of the side-channel resistance of our Ascon S-box designs, we used the formal verification approach by Bloem *et al.* [BGI+18]. The tool is publicly accessible online [Ius18]. The basic idea of this tool is the calculation of the Fourier spectrum (or Walsh transform) at all circuit positions and for all possible signal timings. This Fourier spectrum allows to calculate for each wire which signals leak in the probing model at a specific point in the circuit.

Since the calculation of the exact signal spectrum for all timings is a very complex task, this tool uses a rule-based system to perform an approximation of the spectrum. More specifically, it is approximated which Fourier coefficients are unequal to zero instead of the calculation of concrete values for the coefficients. A nonzero coefficient in the spectrum means that the signal leaks information for this component (a signal or combination of signals). The rules guarantee that the real Fourier spectrum is part of the approximation but does not guarantee to calculate just the nonzero coefficients. In other words, circuits that are accepted by the tool are provably secure for the given input parameters which are the circuit itself and a labeling for the input signals according to three categories (secrets, masks, or public signals). For our side-channel experiments, we verified the low-latency S-box designs of Ascon up to order three. The results are shown in Table 4 (column FV).

It shows that the first-order S-box design is verified in less than two seconds (parallel verification of the five secrets) on our Intel Xeon E5-2699v4 CPU with a clock frequency of 3.6 GHz and 512GB of RAM running in a 64-bit Debian 9 operating system. For order two, the verification increases to about 18 seconds, and for order three the verification takes about 21 minutes. All verification outcomes indicate securely masked circuits for the given protection order.

For the verification of the AES S-box, on the other hand, the circuit size exceeds the number of gates over the most complex circuit tested in the paper of Bloem *et al.* [BGI+18] (a DOM-protected AES S-box verified in 5 to 10 hours) by almost a factor of ten. Therefore, we could not finish the verification within one day and decided to use a verification approach specifically designed for our approach which we refer to as taint checking in the following.

Taint checking of the AES S-box. The basic idea for the taint checking verification approach follows from the design principle of our low-latency masking approach. As already stated in Section 3 in more detail, any $d+1$ masked circuit is trivially secure in the probing model if for any gate or wire of the circuit there is no path that connects any two shares of one variable. We could thus split the circuit into $d+1$ distinct sub-circuits that are never fed by two or more shares of one shared input variable. This approach of course only works if we do not use a share compression like it is the case for the Ascon S-box design and the zero latency AES S-box variant. Other variants of our designs that use the CMS share compression cannot be verified using this approach because the compression clearly creates paths that combine two or more shares (which are then of course first remasked to ensure independence). However, our main goal is to show the security of our low-latency masking approach and to demonstrate that even very complex designs like the AES S-box can be securely implemented this way. The other variants of the AES S-box suggested in Section 6 are introduced to analyze possible trade-offs and implementation costs of our approach.

We instantiated the taint checking approach by using the tool of Bloem *et al.* [BGI+18] as a basis. We label all shared circuit inputs accordingly to the sharing and then simply propagate the input labels through the entire circuit so that every gate and wire that is somehow connected with the input share is tainted by assigning the label of the connected inputs. If at any point in the circuit, two shares from the same variable are part of the labeling of one wire, our tool denotes a flaw and returns the causing gate and inputs. This tool has proven to be extremely helpful also during the design of the AES and Ascon circuits. Therefore (and to make our verification as transparent and accessible as possible), we set up a virtual machine with our taint checking tool and some example circuits [1].

We also performed the taint checking verification approach for the Ascon circuits, for which the verification now takes less than a second. Furthermore, we managed to check the first-order zero latency AES S-box variant in a bit more than ten minutes. We, however, note that this approach works only for this specific kind of low-latency circuits without compression for which the security can be easily verified by ensuring a separation of shares throughout the entire circuit.

Table 4: Side-channel resistance verification results for the low-latency Ascon and the first-order zero latency AES S-box designs

S-box Design	Gates		Order	FV		Taint Checking	
	Lin	Non-lin		Time	Result	Time	Result
1st-order Ascon	34	22	1	$\leq 2\,\text{s}$	✓	$\leq 1\,\text{s}$	✓
2nd-order Ascon	58	48	2	$\leq 18\,\text{s}$	✓	$\leq 1\,\text{s}$	✓
3rd-order Ascon	88	84	3	$\leq 21\,\text{m}$	✓	$\leq 1\,\text{s}$	✓
Zero Latency AES	17,199	5,544	1	≥ 1 day	?	$\leq 11\,\text{m}$	✓

9 Conclusions

In this work, we introduced a generic concept to protect latency constraint applications against side-channel analysis by means of Boolean masking. Our approach works by duplication of the sharing of inputs and circuit parts which hinder an evaluation of the masked circuit in fewer cycles. In addition, we do not perform a share compression step after each nonlinear operation which avoids register stages for the price of an increased share count. We used two case studies based on Ascon and the AES to demonstrate the feasibility of our generic low-latency masking approach. We analyzed the hardware overhead

[1] https://goo.gl/Wph3Ek

and possible trade-offs, and compared our designs to the state of the art. All our designs reduce the amount of latency compared to the existing work. The reduction of latency does not come for free and introduces a significant amount of additional circuitry. However, we showed that even complex circuits like the masked AES S-box can be calculated in a single clock cycle. Furthermore, we showed that the overhead can be traded off against taking compromises in terms of latency. As another interesting aspect, we showed that higher-order masking not necessarily requires online randomness as demonstrated in the zero latency AES S-box.

Future work. Our work raised some interesting research questions towards more efficient low-latency masking in the future. Some aspects were already touched in previous sections. We summarize and extend these questions hereafter:

- Our low-latency approach introduces a significant amount of on-the-fly randomness that is spent on share compression. A straightforward question is thus, how can we leverage the extended domain separation principle to design a less randomness demanding share compression algorithm?
- What are the best design choices like the most suitable AES S-box construction, points in the circuit where to reshare and compress, et cetera, for a given latency constraint?
- The AES S-box inputs which we duplicated to avoid collisions were found using a step-by-step design approach and by concurrently checking and resolving collisions found using our taint checking tool. The duplication, however, is not necessarily optimal. Finding better sharings and proving its optimality remains an open question.
- Finding and resolving collisions in a complex circuit is a very time consuming and cumbersome task. For example multiple collisions in a single gate allow for multiple ways to resolve the collision. The impact of the chosen solution on the overall circuit is not immediately foreseeable. Designing an algorithm that automatically transforms a netlist into a collision-free circuit with the minimum amount of duplications would thus be extremely helpful.

Acknowledgements.

We thank Stefan Mangard for his valuable feedback and for many helpful discussions. The work has been supported in part by the Austrian Science Fund (FWF) through project P26494-N15, project W1255-N23, and S11406. This work has received funding from the European Unions Horizon 2020 research and innovation programme under grant agreement No 644905. The work has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement No 681402).

References

[ABP+17] Victor Arribas, Begül Bilgin, George Petrides, Svetla Nikova, and Vincent Rijmen. Rhythmic Keccak: SCA Security and Low Latency in HW. Cryptology ePrint Archive, Report 2017/1193, 2017. https://eprint.iacr.org/2017/1193.

[BDF+17] Gilles Barthe, François Dupressoir, Sebastian Faust, Benjamin Grégoire, François-Xavier Standaert, and Pierre-Yves Strub. Parallel Implementations of

Masking Schemes and the Bounded Moment Leakage Model. In *EUROCRYPT (1)*, volume 10210 of *Lecture Notes in Computer Science*, pages 535–566, 2017.

[BDN+14] Begül Bilgin, Joan Daemen, Ventzislav Nikov, Svetla Nikova, Vincent Rijmen, and Gilles Van Assche. Efficient and First-Order DPA Resistant Implementations of Keccak. In *CARDIS 2014*, LNCS. 2014.

[BGI+18] Roderick Bloem, Hannes Groß, Rinat Iusupov, Bettina Könighofer, Stefan Mangard, and Johannes Winter. Formal Verification of Masked Hardware Implementations in the Presence of Glitches. EUROCRYPT 2018, 2018.

[BGN+14] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. A More Efficient AES Threshold Implementation. In *AFRICACRYPT*, volume 8469 of *Lecture Notes in Computer Science*, pages 267–284. Springer, 2014.

[BGN+15] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Trade-Offs for Threshold Implementations Illustrated on AES. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 34(7):1188–1200, 2015.

[BP12] Joan Boyar and René Peralta. A Small Depth-16 Circuit for the AES S-Box. In *SEC*, volume 376 of *IFIP Advances in Information and Communication Technology*, pages 287–298. Springer, 2012.

[Can05] David Canright. A Very Compact S-Box for AES. In *CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 441–455. Springer, 2005.

[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*, 2015.

[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*, 2016.

[FWR05] M. Feldhofer, J. Wolkerstorfer, and V. Rijmen. AES Implementation on a Grain of Sand. *IEE Proceedings - Information Security*, 152(1):13–20, Oct 2005.

[GC17] Ashrujit Ghoshal and Thomas De Cnudde. Several Masked Implementations of the Boyar-Peralta AES S-Box. In *INDOCRYPT*, volume 10698 of *Lecture Notes in Computer Science*, pages 384–402. Springer, 2017.

[GM17] Hannes Groß and Stefan Mangard. Reconciling d+1 Masking in Hardware and Software. In *CHES*, volume 10529 of *Lecture Notes in Computer Science*, pages 115–136. Springer, 2017.

[GM18] Hannes Gross and Stefan Mangard. A unified masking approach. *Journal of Cryptographic Engineering*, Mar 2018.

[GMK17] Hannes Groß, Stefan Mangard, and Thomas Korak. An Efficient Side-Channel Protected AES Implementation with Arbitrary Protection Order. In Helena Handschuh, editor, *CT-RSA 2017*, pages 95–112, Cham, 2017. Springer International Publishing.

[Gro18] Hannes Groß. DOM and UMA Masked Hardware Implementations of Ascon. https://github.com/hgrosz/ascon_dom, 2018.

[Gue09] Shay Gueron. Intel's New AES Instructions for Enhanced Performance and Security. In *FSE*, volume 5665 of *Lecture Notes in Computer Science*, pages 51–66. Springer, 2009.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private Circuits: Securing Hardware against Probing Attacks. In *CRYPTO 2003*, volume 2729 of *LNCS*. 2003.

[Ius18] Rinat Iusupov. REBECCA - Masking verification tool. `https://github.com/riusupov/rebecca`, 2018.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. CRYPTO '99, London, UK, 1999. Springer-Verlag.

[MCE07] Edwin NC Mui, R Custom, and D Engineer. Practical implementation of Rijndael S-box using Combinational logic. *Custom R&D Engineer Texco Enterprise Pvt. Ltd*, 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. EUROCRYPT'11. Springer-Verlag, 2011.

[NRR06] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold Implementations Against Side-Channel Attacks and Glitches. In *Information and Communications Security*, volume 4307 of *LNCS*. 2006.

[QS01] Jean-Jacques Quisquater and David Samyde. ElectroMagnetic Analysis (EMA): Measures and Counter-measures for Smart Cards. In *Smart Card Programming and Security*, volume 2140 of *LNCS*. 2001.

[RBN+15] Oscar Reparaz, Begül Bilgin, Svetla Nikova, Benedikt Gierlichs, and Ingrid Verbauwhede. Consolidating Masking Schemes. In *CRYPTO 2015*, 2015.