Efficient Second-Order Masked Software Implementations of Ascon in Theory and Practice

Barbara Gigerl[1], Florian Mendel[2], Martin Schläffer[2], and Robert Primas[3]

[1] Graz University of Technology, Graz, Austria
`first.last@iaik.tugraz.at`
[2] Infineon Technologies, Munich, Germany
`first.last@infineon.com`
[3] Intel Labs, Hillsboro, USA
`first.last@intel.com`

Abstract. In this paper, we present efficient protected software implementations of the authenticated cipher Ascon, the recently announced winner of the NIST standardization process for lightweight cryptography. Our implementations target theoretical and practical security against second-order power analysis attacks.
First, we propose an efficient second-order extension of a previously presented first-order masking of the Keccak S-box that does not require online randomness. The extension itself is inspired by a previously presented second-order masking of an And-Xor construction. We then discuss implementation tricks that further improve performance and reduce the chance of unintended combination of shares during the execution of masked software on microprocessors. This allows us to retain the theoretic protection orders of masking in practice with low performance overhead, which we also confirm via TVLA on ARM microprocessors. The formal correctness of our designs is additionally verified using Coco on the netlist of a RISC-V Ibex core.
We benchmark our masked software designs on 32-bit ARM and RISC-V microprocessor platforms. On both platforms, we can perform Ascon-128 authenticated encryption with a throughput of about 300 or 550 cycles/byte when operating on 2 or 3 shares. When utilizing a leveled implementation technique, the throughput of our masked implementations generally increases to about 90 cycles/byte.
We publish our masked software implementations together with a generic software framework for evaluating performance and side-channel resistance of various masked cryptographic implementations.

Keywords: Ascon, Software, Masking, Verification

1 Introduction

Implementation attacks such as fault attacks [17,13,26] or power analysis [42,51,18] are among the most relevant threats for implementations of cryptographic algorithms. To counteract such attacks, cryptographic devices like smart cards typically implement dedicated countermeasures on algorithmic level. The most prominent examples of algorithmic countermeasures are masking against power analysis [55,45,53,39], and the usage of some form of redundancy against fault attacks [4,22].

Masking is a secret-sharing technique that splits up cryptographic computations in multiple shares that, when observed individually, do not reveal any useful information about the processed data. This technique can be used to counteract power analysis techniques such as differential power analysis (DPA) [42].

Redundant computations are usually used to detect and prevent the release of erroneous cryptographic computations, that could otherwise be exploited by an attacker that physically tampers with the device, using techniques like differential fault attacks [17] or statistical fault attacks [31,26].

One of the main practical challenges with implementation security is the accompanied overhead, in terms of area/code size and runtime, that can increase by several orders of magnitude compared to plain (unprotected) implementations [11], mostly due to the overhead of masking countermeasures. The importance of efficiency is also reflected by the NIST standardization process for lightweight cryptography [46] that recently came to a close. Here, the goal was to select future standards for authenticated encryption that should not only outperform current AES-based schemes but also, amongst others, allow the addition of countermeasures against implementation attacks at low cost.

One way to achieve efficient protected implementations is to use cryptographic schemes based on so-called lightweight building blocks that are comparably cheap to protect against implementation attacks. Nearly all candidates in the final round of the standardization process follow this approach, e.g., by using cryptographic building blocks with low-degree nonlinear layers that keep the overhead of masking comparably low. On top of that, efficiency can be further improved by using of cryptographic modes that can either reduce the attack surface of certain implementation attacks or prevent them entirely. The attack surface of DPA-based key recovery attacks can be reduced, e.g., by using cryptographic modes allowing so-called leveled implementations that restrict the need for algorithmic countermeasures to only certain parts of a cryptographic computation [50,28]. Ascon, the winner of the standardization process for lightweight cryptography, follows both of these approaches.

Related Work Besides mode-level properties, another way to improve the efficiency of protected cryptographic implementations is to design more efficient algorithmic countermeasure techniques. These efforts mostly focus on optimizing masking countermeasures as their performance overhead scales quadratically with the desired security level. One important optimization goal for masking countermeasures is to reduce the amount of randomness needed during the execution of a masked cryptographic algorithm, which decreases the cost of additionally required RNGs. In recent years, multiple works have already proposed efficient masking schemes for various (symmetric) cryptographic algorithms requiring low online randomness [61,12,22,57,58]. Many of these works consider masked implementations of cryptographic algorithms in hardware, often at the cost of increasing the area of the cryptographic hardware circuit.

In software, however, such trade-offs are often not desirable as increasing the computation state, i.e., the number of temporary variables required for computation, also increases software runtime. In fact, increasing the computation state often disproportionally increases software runtime since a processor will need to make excessive use of load and store instructions to keep all currently required temporary variables in the register file [34]. Hence, efficient masked software needs find good trade-offs between the size of the computation state and the amount of required randomness from hardware/software RNGs. On top of that, another practically relevant problem is side-effects of processor microarchitectures (glitches, transitions) that cause reductions in masking security orders for many published masked software implementations [9].

Independently to this work, Gaspoz et al. published a work on threshold implementations in software that also takes micro-architectural leakage effects into consideration [32]. To achieve, amongst others, non-completeness with respect to architectural leakages in the register file, they propose storing certain sets of intermediate (shared) values with certain rotation offsets. We make use of a similar idea to (1) reduce the amount of fresh randomness for our constructions, (2) harden the resulting implementations against various architectural leakages. The authors of [32] also acknowledge the concurrent proposal of this idea by citing a codebase containing a previous version of the masked Ascon implementations presented in this paper.

Our Contribution

- We present efficient software implementations of Ascon-128 that come with theoretical and practical security against second-order power analysis attacks. Our designs do not require any online randomness which makes them particularly efficient on low-end devices. While we do use the Ascon cipher as the main discussion example, we also explain how ideas can be applied to many other lightweight symmetric ciphers.
- We present additional implementation techniques that (1) further improve performance and (2) help masked software implementations retain their theoretical protection order while being executed on real microprocessors, a problem that is particularly widespread amongst published masked software implementations.
- We benchmark our masked software designs on common 32-bit ARM and RISC-V microprocessor platforms.
- We verify the practical and theoretical correctness of our masked implementations using TVLA on a STM32F3 32-bit ARM microprocessor, as well as formal verification using Coco and the netlist of the 32-bit RISC-V Ibex core.
- We build a generic software framework based on the chipwhisperer toolchain that allows practical evaluations of masked cryptographic software. The

framework has convenient features such as automatic sharing of data during transmission over the serial interface, optional external bitinterleaving or endian-swaps, features benchmark and TVLA scrips, and supports arbitrary orders of cipher input arguments. The code of our masked software implementations and analysis framework is available on github[1].

Outline In Section 2, we cover preliminaries on the authenticated cipher Ascon and power analysis countermeasures. Section 3 explains the design of our masked software implementations of Ascon. Section 4 describes how we use the tool Coco to verify the formal correctness of our masked software designs. Section 5 describes performance metrics and practical evaluation results of our implementations. We conclude our work in Section 6.

2 Background

2.1 Ascon

The cipher suite Ascon provides authenticated encryption with associated data and hashing functionality, and has recently been selected as the new standard for lightweight cryptography in the NIST Lightweight Cryptography competition [47]. The Ascon suite consists of the authenticated ciphers Ascon-128 and Ascon-128a, the hash functions Ascon-Hash and Ascon-Hasha, and the extendable output functions Ascon-Xof and Ascon-Xofa. All schemes provide 128-bit security and internally use the same permutation Ascon-p operating on a 320-bit state that is organized into 5×64 bit lanes. Ascon-p consists of 3 steps: a round constant addition, a non-linear substitution layer, and a linear mixing layer, that are consecutively applied on the state in each round (for details see Section 6).

Ascon's modes describe how Ascon-p can be used to realize authenticated encryption, hashing, or extendable output functions. For the purpose of this paper, we only give descriptions of the authenticated encryption schemes Ascon-128 and Ascon-128a that are of main interest in the context of implementation attacks. Here, the input consists of a secret key K, a nonce N, associated data A, and a plaintext P. The produced output consists of the authenticated ciphertext C plus an authentication tag T, which authenticates both the associated data and the encrypted message. The decryption and verification procedure takes as input the key K, nonce N, associated data A, ciphertext C and tag T, and outputs either the plaintext P if the verification of the tag is correct or an error if the verification of the tag fails. Figure 1 illustrates the authenticated encryption modes of the Ascon suite. Table 1 contains additional parameters of these modes. The sizes of associated data A and plaintext P are arbitrary, the ciphertext C has the exact same length as P.

In the context of implementation security, one especially interesting property of the Ascon mode is its keyed initialization and finalization (indicated in

[1] https://github.com/ascon/simpleserial-ascon

blue in Figure 1) which protects against trivial key recovery and forgery attacks even if an attacker somehow gets knowledge of an internal state during the data procession of Ascon. This property hence allows for so-called leveled implementations where the degree of algorithmic countermeasures can be reduced during the data processing phase to improve efficiency [50,28].

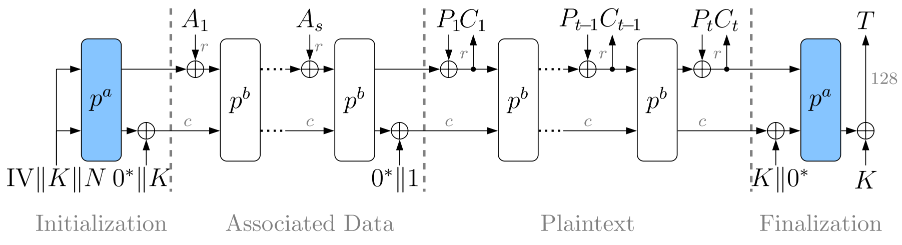


Fig. 1: Illustration of Ascon's mode for authenticated encryption. Protection against DPA-based key recovery attacks can be achieved by only adding algorithmic countermeasures to the initialization and finalization phase (indicated in blue).

Table 1: Recommended parameters for Ascon's modes for authenticated encryption.

Name	Bit size of					Rounds	
	K	N	T	r	c	a	b
Ascon-128	128	128	128	64	256	12	6
Ascon-128a	128	128	128	128	192	12	8

2.2 Masking

Masking is an algorithmic countermeasure against power analysis attacks such as differential power analysis [42]. In a nutshell, masking is a secret-sharing technique that splits intermediate values of a computation into $d+1$ uniformly random shares, such that observing up to d shares does not leak any information about the underlying value. The used masking scheme determines the number of masks d, and results in a dth-order masking scheme. In classical Boolean masking, the sharing of a native variable s, when split into $d+1$ random shares $s_0 \ldots s_d$, must satisfy $s = s_0 \oplus \ldots \oplus s_d$. Hereby, $s_0 \ldots s_{d-1}$ is chosen uniformly at random while $s_d = s_0 \oplus \ldots \oplus s_{d-1} \oplus s$. This ensures that each share s_i is uniformly distributed and statistically independent of s. For example, in a first-order masking scheme ($d = 1$), the secret variable s is split up into two shares s_0 and s_1, such that $s = s_0 \oplus s_1$. s_0 is chosen uniformly at random, while $s_1 = s \oplus s_0$.

When implementing masked cryptographic algorithms, dealing with linear functions is trivial as they can simply be computed on each share individually. However, implementing masking for non-linear functions requires computations on all shares of a native value, which is more challenging to implement in a secure and correct manner, and thus the main interest in literature [40,45,54,38,10,22,33,58].

2.3 Formal Verification of Masking

Masked implementations generally need to take care that each intermediate variable of a computation is statistically independent of any native (unmasked) values. The verification of this property is usually done with the help of a security model that specifies the abilities of an attacker. Typically, it is assumed that the ability of the attacker is to place a certain amount of probes in a computation, that allow monitoring concrete values at those locations.

The *classical probing model* by Ishai et al. [40] is the most commonly used security model for masked hardware circuits and it's accuracy in modeling real world attacks has been confirmed by many works [30,56,38,33,58]. Here, an attacker is allowed to place up to d probes at any location in a circuit, which can be used to observe the corresponding gates/wires permanently. A masked hardware circuit is considered d^{th}-order secure if an attacker cannot learn any information about the native values by combining all d observations. Examples of tools that can verify classical probing security for masked hardware circuits are Rebecca [15], `Silver` [41], and `maskVerif` [5].

On software side, there exist many methods and tools for automatically generating or verifying masked software implementations [7,44,8,29,62,6,10]. These tools model an attacker to place probes on individual words of a processor's register file, and to use them for one cycle each during the execution of a masked software implementation. Hereby, it is assumed that the probed registers cause independent leakage, in other words, no additional potential side effects of a processors architecture, such as glitches or register overwrites, are considered [52]. With Coco, Gigerl et al. have presented a tool that can verify the correctness of masked software implementations while considering possible architectural side effects of a given processor netlist [33].

2.4 Coco

Coco is a tool for the co-design and co-verification of masked software implementations on processor netlists [33]. Coco formally verifies the security of (any-order) masked assembly implementations that are executed on concrete processors, defined by gate-level netlists.

Coco considers the *time-constrained probing model*, which allows an attacker to distribute d probes in the processor netlist, in arbitrary execution cycles of the masked software. Each probe can be used to measure information in one specific clock cycle and at one specific location. The attacker can distribute the d probes spatially and temporally. Hence, the attacker can perform d measurements at different locations in the same clock cycle, or probes at the same location in different clock cycles, or a mix of both. A masked software implementation is considered dth-order secure in the time-constrained probing model if an attacker cannot combine the recorded information to learn anything about native variables.

This security notion is verified in Coco by (1) defining an initial labeling that indicates the location and dependencies of shares in a processor netlist prior to the start of masked software execution, (2) propagating these labels efficiently encoded as correlation sets throughout the netlist until the execution of masked software is finished. In a nutshell, a correlation set contains the labels of all variables which might be visible to the attacker on the gate output during a clock cycle. For example, an Xor gate with 1-bit share a, and 1-bit random variable r, as inputs will generate the correlation set $\{a \oplus r\}$ if we only consider stable signals or $\{0, a, r, a \oplus r\}$ if we additionally consider glitches due to propagation delay of signals. Put differently, an attacker can either observe an arbitrary independent constant (denoted by 0), the share a (if r is delayed), the randomness r (if a is delayed), or $a \oplus r$ once the circuit has stabilized. In contrast, an And gate will generate the correlation set $\{0, a, r, ar\}$ even if we only consider stable signals [15,33]. Coco reports a leak if there exists a correlation set in the circuit which contains a term which directly depends on the native (unmasked) variables in any clock cycle. For dth-order masking verification, Coco will check if any combination of up to d probes depends on native variables.

One additional outcome of the work in [33] is a modified, *secured* version of the Ibex core. This secured Ibex core features several small adaptions of the microarchitecture that eliminate various sources of masking-related side-channel leakage that are otherwise hard, if not impossible, to compensate for purely in software. Additionally, they state a couple of constraints to be followed by masked software that are otherwise to costly to address in hardware entirely. These constraints mainly boil down to (1) shares of the same native value must not be accessed within two successive instructions, and (2) a register or memory location containing one share must not be overwritten by another share of the same native value.

3 Protected Software Implementations of Ascon

In this section, we describe efficient masking schemes for Ascon in software. We mostly focus our discussion on the 5-bit χ S-box, which is prominently used in Keccak and in the core of the Ascon S-box (cf. Section 6). First, we recall a previously presented efficient 2-share masking scheme for χ that serves as the basis for our designs. We then describe how we extend this design using 3 shares to lift probing security to the second order at low cost. After that, we describe how we design an entire round of Ascon-p using additional implementation tricks that further improve performance and reduce the impact of glitches/transitions on microprocessors. Finally, we discuss how our masking scheme can also be applied to software implementations of other cryptographic algorithms.

3.1 2-share Design of the χ S-box

We now recall the 2-share design of the 5-bit χ S-box from Daemen et al. [22] that is based on ideas from Sugawara et al. [61] and Vivek et al. [60]. We denote the input bits of χ with a, b, c, d, e plus an additional intermediate variable r. The shared versions of these variables are indicated with subscripts. The 2-share design χ_{2S} relies on repeated calls of the 2-share Toffoli gate $p_{\chi 2S}$:

```
Name: $\chi_{2S}$
In-/Output: $\{a_0, a_1, b_0, b_1, c_0, c_1, d_0, d_1, e_0, e_1, r_0, r_1\}$
$p_{\chi 2S}(r_0, r_1, e_0, e_1, a_0, a_1)$
$p_{\chi 2S}(a_0, a_1, b_0, b_1, c_0, c_1)$
$p_{\chi 2S}(c_0, c_1, d_0, d_1, e_0, e_1)$
$p_{\chi 2S}(e_0, e_1, a_0, a_1, b_0, b_1)$
$p_{\chi 2S}(b_0, b_1, c_0, c_1, d_0, d_1)$
$d_0 \leftarrow d_0 \oplus r_0$
$d_1 \leftarrow d_1 \oplus r_1$
```

Construction 3.1: 2-share χ S-box from [22].

```
Name: $p_{\chi 2S}$
In-/Output: $\{c_0, c_1, a_0, a_1, b_0, b_1\}$
$c_0 \leftarrow c_0 \oplus \overline{a_0} b_1$
$c_0 \leftarrow c_0 \oplus \overline{a_0} b_0$
$c_1 \leftarrow c_1 \oplus a_1 b_1$
$c_1 \leftarrow c_1 \oplus a_1 b_0$
```

Construction 3.2: 2-share Toffoli gate from [22].

The 2-share Toffoli gate $p_{\chi 2S}$, here used with an additionally negated input, takes as input the shares of a, b, c and calculates $c \leftarrow c \oplus \overline{a}b$. $p_{\chi 2S}$ is comprised of four calls of the ordinary Toffoli gate in succession, each of which receives the updated variables of previous calls and operates on an incomplete set of shares. Since each ordinary Toffoli gate is invertible, given the construction of $p_{\chi 2S}$, it is possible to directly calculate input shares from output shares, which makes $p_{\chi 2S}$ invertible and free of entropy loss.

As described in [60], every permutation (S-box) with an odd number of inputs can be implemented using reversible (Toffoli) gates by using at most one additional variable. χ_{2S} is a masked variant of one such implementation. The additional masked input variables r_0 and r_1 of χ_{2S} should be initialized such that they represent a sharing of zero, i.e., $r_0 \oplus r_1 = 0$. Since χ_{2S} is a permutation on the input shares, it is possible to use one share r_0 of the output of one S-box layer as input to the next layer of S-boxes without reducing the entropy of the state. Hence, it is possible to implement entire masked ciphers without the need for additional online randomness, except the one needed for r_0 and r_1 in the initial sharing of the first S-box layer.

Besides the suitability for masking, χ_{2S} has another convenient property as it can be combined with redundant computation to achieve protection from statistical ineffective fault attacks (SIFA) that are otherwise notoriously difficult to defend against [22]. More concretely, the χ_{2S} construction ensures that, within one S-box computation, a single fault induction either (1) cannot cancel out based on the all shares of any native value, or (2) is detectable by comparing the result of the S-box computation to a redundant computation that is typically needed anyway to cope with other fault attacks. Put differently, if a fault induction causes a difference within one of the ordinary Toffoli gates, it can only cancel out due to an AND gate computation on incomplete sets of shares. If a fault induction causes a difference outside of an ordinary Toffoli gate, it will propagate to the S-box output where it can be detected by comparison with a redundant computation.

3.2 3-share Design of the χ S-box

The main idea behind our 3-share design of χ is to keep the general structure of the 2-share design but to use 3-share Toffoli gates instead. While this is not difficult in principle, the situation becomes more challenging if we additionally want to avoid using any online randomness. In this context, a recent work from Shahmirzadi et al. [58] has explored the possibilities of implementing various quadratic functions such as χ with second-order probing security and without the requirement of online randomness. Their constructions are based on the AND-XOR$_{3S}$ construction that calculates $x \leftarrow ab + c$ on three shares:

As later stated in their paper, while AND-XOR$_{3S}$ produces correct outputs, the computation itself is not second-order probing secure. Nevertheless, we use AND-XOR$_{3S}$ as the basis for designing $p_{\chi 3S}$, a Toffoli gate that calculates $c \leftarrow c \oplus \overline{a}b$ on 3 shares.

From a runtime perspective, the main benefit of our Toffoli gate construction is that it allows expressing the χ S-box as a sequence of permutations which reduces the computation state, i.e., the number of temporary variables required for computation. This benefits software runtime, especially on low-end devices with limited register file sizes, and with increasing masking order. Besides that, our construction also features some logic-level optimizations that further reduce the amount of computation steps.

In any case, additional measures need to be taken to make both of these constructions second-order probing secure. The main problem with our second-order extension of the masked Toffoli gate is that the single c term is no longer sufficient to refresh the multiplication of a and b if two probes can be used by an attacker. Hence, if we want to implement $p_{\chi 3S}$ as a permutation on the input shares, we need to increase the number of inputs by introducing the additional refreshing terms $\mathrm{R}_0, \mathrm{R}_1, \mathrm{R}_2$ representing a sharing of zero. These terms can then be used for refreshing in the individual calls of $p_{\chi 3S}$. The description of the 3-share χ_{3S} is given in Construction 3.5.

```
Name: And-Xor$_{3S}$
Input:$\{a_0, a_1, a_2, b_0, b_1, b_2, c_0, c_1, c_2\}$
Output:$\{x_0, x_1, x_2\}$
$x'_0 \leftarrow a_0b_0 \oplus b_0$
$x'_1 \leftarrow a_0b_1$
$x'_2 \leftarrow a_0b_2 \oplus c_0$
----------------------------------------
$x'_3 \leftarrow a_1b_0 \oplus b_0$
$x'_4 \leftarrow a_1b_1$
$x'_5 \leftarrow a_1b_2 \oplus b_2 \oplus c_1$
----------------------------------------
$x'_6 \leftarrow a_2b_0 \oplus a_2 \oplus c_2$
$x'_7 \leftarrow a_2b_1$
$x'_8 \leftarrow a_2b_2 \oplus a_2 \oplus b_2$
----------------------------------------
$x_0 \leftarrow x'_0 \oplus x'_1 \oplus x'_2$
$x_1 \leftarrow x'_3 \oplus x'_4 \oplus x'_5$
$x_2 \leftarrow x'_6 \oplus x'_7 \oplus x'_8$
```

Construction 3.3: 3-share And-Xor from [58].

```
Name: $p_{\chi 3S}$
In-/Output:$\{c_0, c_1, c_2, a_0, a_1, a_2, b_0, b_1, b_2,$
$\mathrm{R}_0, \mathrm{R}_1, \mathrm{R}_2\}$
$c_0 \leftarrow c_0 \oplus a_0b_2$
$c_0 \leftarrow c_0 \oplus a_0b_1 \oplus \mathrm{R}_2$
$c_0 \leftarrow c_0 \oplus \overline{a_0}b_0$
----------------------------------------
$c_1 \leftarrow c_1 \oplus a_1b_2$
$c_1 \leftarrow c_1 \oplus \overline{a_1}b_1 \oplus \mathrm{R}_0$
$c_1 \leftarrow c_1 \oplus a_1b_0$
----------------------------------------
$c_2 \leftarrow c_2 \oplus \overline{b_0}a_2$
$c_2 \leftarrow c_2 \oplus a_2b_1 \oplus \mathrm{R}_1$
$c_2 \leftarrow c_2 \oplus a_2 \obar b_2$
----------------------------------------
```

Construction 3.4: Our 3-share Toffoli gate (with negated a).

```
Name: $\chi_{3S}$
In-/Output:
$\{a_0, a_1, a_2, b_0, b_1, b_2, c_0, c_1, c_2, d_0, d_1, d_2, e_0, e_1, e_2,$
$r_0, r_1, r_2, \mathrm{R}_0, \mathrm{R}_1, \mathrm{R}_2\}$
$p_{\chi 3S}(r_0, r_1, r_2, e_0, e_1, e_2, a_0, a_1, a_2, \mathrm{R}_0, \mathrm{R}_1, \mathrm{R}_2)$
$(\mathrm{R}_0, \mathrm{R}_1, \mathrm{R}_2) \leftarrow (\mathrm{R}_0, \mathrm{R}_1, \mathrm{R}_2) \ggg 1$
$p_{\chi 3S}(a_0, a_1, a_2, b_0, b_1, b_2, c_0, c_1, c_2, \mathrm{R}_0, \mathrm{R}_1, \mathrm{R}_2)$
$(\mathrm{R}_0, \mathrm{R}_1, \mathrm{R}_2) \leftarrow (\mathrm{R}_0, \mathrm{R}_1, \mathrm{R}_2) \ggg 1$
$p_{\chi 3S}(c_0, c_1, c_2, d_0, d_1, d_2, e_0, e_1, e_2, \mathrm{R}_0, \mathrm{R}_1, \mathrm{R}_2)$
$(\mathrm{R}_0, \mathrm{R}_1, \mathrm{R}_2) \leftarrow (\mathrm{R}_0, \mathrm{R}_1, \mathrm{R}_2) \ggg 1$
$p_{\chi 3S}(e_0, e_1, e_2, a_0, a_1, a_2, b_0, b_1, b_2, \mathrm{R}_0, \mathrm{R}_1, \mathrm{R}_2)$
$(\mathrm{R}_0, \mathrm{R}_1, \mathrm{R}_2) \leftarrow (\mathrm{R}_0, \mathrm{R}_1, \mathrm{R}_2) \ggg 1$
$p_{\chi 3S}(b_0, b_1, b_2, c_0, c_1, c_2, d_0, d_1, d_2, \mathrm{R}_0, \mathrm{R}_1, \mathrm{R}_2)$
$d_0 \leftarrow d_0 \oplus r_0$
$d_1 \leftarrow d_1 \oplus r_1$
$d_2 \leftarrow d_2 \oplus r_2$
```

Construction 3.5: Our 3-share second-order secure χ S-box.

While each call of $p_{\chi 3S}$ requires independent refreshing terms, we can achieve this by simply rotating each term by some constant to derive a new refreshing term for the next $p_{\chi 3S}$ call. In the above description, this is denoted as $(\mathrm{R}_0, \mathrm{R}_1, \mathrm{R}_2) \ggg 1$ for a rotation offset of one. With this trick, our construction again becomes a permutation of shares, although, at the expense of an increased computation state.

3.3 Further Performance Improvements and SCA-Hardening

We now discuss additional steps that can be taken to improve the performance and practical side-channel resistance of our masked software implementations. We then describe how we implement the round function Ascon-p, the main building block of our Ascon implementations that will be used in the later sections for benchmarks and formal/empirical masking verification.

The most notable downside of our previously discussed χ_{3S} construction, when compared to χ_{2S}, is the increase of the computation state with refreshing terms which causes increased register spilling and thus performance degradation. If we allow ourselves to deviate a bit from the so-far used design strategy, it is however still possible to avoid most of this overhead. More concretely, we can replace the previously required refreshing terms $\mathrm{R}_0, \mathrm{R}_1, \mathrm{R}_2$ with rotated versions of the already existing additional inputs r_0, r_1, r_2 that also represent a shared zero. On top of that, and to avoid potential entropy loss of the state over multiple rounds, we can then add these (rotated) refreshing terms back to the state towards the end of the S-box computation. If we apply these modification to our χ_{3S} construction, we end up with the optimized $\chi_{3S^\star}$ variant in Construction 3.6.

The main consequence of our modifications is that $\chi_{3S^\star}$ is not a direct permutation of shares anymore since the values of $\mathrm{R}_0, \mathrm{R}_1, \mathrm{R}_2$ are not part of the input/output anymore. From a masking perspective, however, the computation of one round of $\chi_{3S^\star}$ is still correct since the refreshing terms are still in derived from other independent computations in a changing of the guards fashion [21]. While this argument does not necessarily imply the correctness of masking over multiple rounds, we do show in a later practical evaluation that we do not observe any degradation in masking protection order over multiple rounds (cf. Section 5.2).

Nevertheless, one property that may be lost is SIFA protection if this construction is combined with redundant computation. More concretely, a fault induction outside of a Toffoli gate may not always propagate to the S-box output.

SCA Hardening So far, we have discussed our masked constructions in a somewhat abstract probing model. If one now wants to map these constructions into concrete software implementations, one needs to additionally consider that the practical security of masked software implementations does depend on some assumptions that may not be satisfied when they are being executed on real processors. Coron et al. [19] were among the first who showed that, e.g.,

```
Name: $\chi_{3S^\star}$
In-/Output:
$\{a_0, a_1, a_2, b_0, b_1, b_2, c_0, c_1, c_2, d_0, d_1, d_2, e_0, e_1, e_2,$
$r_0, r_1, r_2\}$
$(\mathrm{R}_0, \mathrm{R}_1, \mathrm{R}_2) \leftarrow (r_0, r_1, r_2) \ggg 1$
$p_{\chi 3S}(r_0, r_1, r_2, e_0, e_1, e_2, a_0, a_1, a_2, \mathrm{R}_0, \mathrm{R}_1, \mathrm{R}_2)$
$(\mathrm{R}_0, \mathrm{R}_1, \mathrm{R}_2) \leftarrow (\mathrm{R}_0, \mathrm{R}_1, \mathrm{R}_2) \ggg 1$
$p_{\chi 3S}(a_0, a_1, a_2, b_0, b_1, b_2, c_0, c_1, c_2, \mathrm{R}_0, \mathrm{R}_1, \mathrm{R}_2)$
$(\mathrm{R}_0, \mathrm{R}_1, \mathrm{R}_2) \leftarrow (\mathrm{R}_0, \mathrm{R}_1, \mathrm{R}_2) \ggg 1$
$p_{\chi 3S}(c_0, c_1, c_2, d_0, d_1, d_2, e_0, e_1, e_2, \mathrm{R}_0, \mathrm{R}_1, \mathrm{R}_2)$
$(\mathrm{R}_0, \mathrm{R}_1, \mathrm{R}_2) \leftarrow (\mathrm{R}_0, \mathrm{R}_1, \mathrm{R}_2) \ggg 1$
$p_{\chi 3S}(e_0, e_1, e_2, a_0, a_1, a_2, b_0, b_1, b_2, \mathrm{R}_0, \mathrm{R}_1, \mathrm{R}_2)$
$(\mathrm{R}_0, \mathrm{R}_1, \mathrm{R}_2) \leftarrow (\mathrm{R}_0, \mathrm{R}_1, \mathrm{R}_2) \ggg 1$
$p_{\chi 3S}(b_0, b_1, b_2, c_0, c_1, c_2, d_0, d_1, d_2, \mathrm{R}_0, \mathrm{R}_1, \mathrm{R}_2)$
$(\mathrm{R}_0, \mathrm{R}_1, \mathrm{R}_2) \leftarrow (\mathrm{R}_0, \mathrm{R}_1, \mathrm{R}_2) \ggg 1$
$r_0 \leftarrow r_0 \oplus \mathrm{R}_0$
$r_1 \leftarrow r_1 \oplus \mathrm{R}_1$
$r_2 \leftarrow r_2 \oplus \mathrm{R}_2$
$d_0 \leftarrow d_0 \oplus r_0$
$d_1 \leftarrow d_1 \oplus r_1$
$d_2 \leftarrow d_2 \oplus r_2$
```

Construction 3.6: Our optimized 3-share second-order secure χ S-box. The main modifications compared to χ_{3S} are highlighted in blue.

memory transitions in the register file or RAM can leak the Hamming distance between two shares, thereby reducing the protection order of masking schemes on processors. Later publications follow up on these observations and describe many more potential sources for the reduction of security order of masked software implementations due transition or glitch effects in processor microarchitecture [49,37,43,2,33,35]. Consequently, if one wants to counteract such problems, one can either use masking scheme with a protection order that is higher than theoretically required, or employ additional hardening tricks in software. We opted to go with the second option.

Our main hardening technique involves the usage of constant rotation offsets between all the shares of a native variable. This reduces information leakage in the case that an unintentional combination of shares occurs in the processor microarchitecture. The rotation offsets can be chosen in a way such that their greatest common divisor is high, which reduces the information leakage, e.g., if the hamming distance between two rotated shares is observed. Naturally, whenever a computation needs to be performed on multiple shares of the same native variable, they need to be rotated back temporarily. For our purposes, we decided on using the rotation offsets 0, 5, and 10 for our 3-share implementation.

Besides that, we also add to our implementations the optional possibility of transmitting inputs and outputs of our Ascon implementations in a shared representation, with or without bit-interleaving, and with or without swapped endianness. This further reduces the processing of masked inputs/outputs which can present another source of leakage in a practical side-channel evaluation.

The Design of Ascon-*p* Given masked descriptions of χ, a protected implementation of the entire round function Ascon-p is not too much additional work. The Ascon S-box can be viewed as χ with additional affine layers at the beginning and the end (cf. Section 6). These affine layers can be treated similar as linear layers, i.e., they are computed individually on each share of the state. The bitsliced algorithmic description of χ allows to split its computation into multiple parts, e.g., computation on the low and high words of the lanes in case of 32-bit implementations. What remains is the linear layer which calculated on each lane of each share individually. We keep the entire state in a bitinterleaved representation such that 64-bit rotations can be more efficiently implementation if dedicated 32-bit rotation instructions are available.

3.4 Application to Other Cryptographic Algorithms

While we only discuss masking techniques for software implementations of Ascon (or Keccak), these techniques are also applicable to S-box layers of many other cryptographic block ciphers or permutations.

3-bit S-boxes have recently become prominent, e.g., with their usage in LowMC [1], or Xoodoo [23]. We focus our discussion on the 3-bit χ-layer of Xoodoo [20,23]. Daemen et al. pointed out that it is possible to compute 3-bit χ in-place in its registers as a sequence of three Toffoli gates [23] which makes it compatible with the optimizations techniques presented in this section. Toffoli-based descriptions for other affine equivalent 3-bit S-boxes can be derived by finding corresponding affine layers using tools such as PEIGEN [3].

4-bit S-boxes have been partitioned into 302 equivalence classes by De Cannière [25] where one class contains all affine functions, six classes contain quadratic functions, and 295 classes represent the cubic functions. As shown by Bilgin et al. [14], 144 cubic classes can be constructed by iterating the S-boxes of the quadratic classes separated by affine layers up to 3 times. This covers many prominent S-boxes, e.g., the S-boxes used in Noekeon [24] and Present [16]. Toffoli-based descriptions for all quadratic 4-bit S-boxes are given in [22].

While we already discuss Toffoli-based descriptions for the 5-bit Ascon and Keccak S-boxes in this paper, such descriptions still need to be found for many other larger S-boxes. One exception is the 8-bit AES S-box with a Toffoli-based description in [22], however, it should be noted that this description is primarily intended for hardware designs as it will not allow efficient execution in software.

4 Formal Masking Verification

In this section, we describe how we apply the formal verification tool Coco [33] to verify the correctness of our masked designs of χ and Ascon-p from the previous section. We first describe the general verification flow of Coco and how we adapt it for our purposes. We then discuss the verification results.

4.1 Verification Flow

We use the secured version of the 32-bit Ibex core from [33] as the reference processor netlist for our formal verification of masked software designs. This core is roughly comparable to an ARM Cortex-M0 in terms of area and performance.

As a first step for the verification process, we prepare RISC-V assembly implementations of the previously presented masked designs of Ascon-p that adhere to all constraints for masked software listed in [33] to achieve protection against transitions and glitches on the secured Ibex core. We then copy the assembly code into the SRAM model that is used by the netlist simulation of the secured Ibex core. Next, we assign labels indicating the position and dependencies of shares at the start of the execution of the masked software. Labels are either *shares* of a native value, *fresh randomness* in case of a fresh independent random variable, or *public*, which includes constants and control signals like the clock signal. In our case, we simply label the contents of the register file which holds the shares of the Ascon-p state before the start of the computation. For the 2-share implementation based on χ_{2S} we label the entire masked state and execute one round of Ascon-p. For the 3-share implementation based on $\chi_{3S^\star}$ we only add labels for the lower 32-bit of each lane since the Ibex register file (32×32-bit) cannot hold 3 shares of the entire Ascon-p state at once. During verification we then execute one entire round of Ascon-p in case of the 2-share implementation, and the Ascon S-box for the lower word of each lane in case of the 3-share implementation. While we could also include the computation for the upper words in the verification of our 3-share implementation, e.g., by executing them one after another and loading/storing them in SRAM, we avoid this step since they are anyway independent from each other.

During the verification with Coco, these labels are propagated through the netlist until the execution of the masked software implementation is finished. Coco reports a leak if there exists a correlation set in the circuit which contains a term which directly depends on the native (unmasked) value. In case of second-order masking verification, Coco will check if any combination of up to two probes depends on native variables.

4.2 Verification Results

We have summarized our verification results, as well as the corresponding runtime of the verification procedures in Table 2. Stable verification only considers probes of signals in the processor netlist after they have stabilized while transient verification also considers side-effects such as transitions and glitches in the netlist.

In case of our 2-share implementation, we could immediately successfully verify first-order security for stable and transient masking verification. The verification runtime for transient execution is significantly increased which is mainly due to the large amount of possible glitches during rotation operations in the linear layer.

In case of our 3-share implementation, we could not immediately verify second-order security successfully. As it turns out, the Ibex core, same as similar other processors, feature logic in the ALU for the computation of sign-bits and processor flags that does cause masking-related issues with our $\chi_{3S^\star}$ construction. More concretely, while our used rotation offset by one should result in independent refreshing terms in theory, the additional ALU logic does violate this assumption. Nevertheless, a simple increase of the rotation offset from one to two eliminates this problem on this core and we can successfully verify second-order security for stable and transient masking verification. This time, the runtime between stable and transient masking verification is smaller since only the S-box execution is considered in the verification.

Implementation	Input Labels	Order	Stable		Transient	
			Result	Time	Result	Time
2-share Ascon-p round	$5 \times 64 \times 2$ bits	1	✓	3m	✓	5h 20m
3-share Ascon S-box	$5 \times 32 \times 3$ bits	2	✓	26m	✓	1h 17m

Table 2: Summary of formal masking verification results on the secured Ibex core. Verification runtimes stem from single-threaded executions on an Intel Core i7 notebook processor with 16GB of RAM.

5 Performance and Side-Channel Evaluation

In this section, we present performance numbers, as well as practical side-channel evaluation results of our masked software implementations. We first compare the performance of Ascon-128 implementations using 2 shares (χ_{2S}) and 3 shares ($\chi_{3S^\star}$) to plain (unmasked) implementations. We then evaluate practical first and second-order security of our masked implementations using test-vector leakage assessment (TVLA) methodology.

5.1 Performance Evaluation

STM32F3 For our performance evaluation on ARM microprocessors, we use a `STM32F303` microprocessor as target devices[4]. This device is based on the 32-bit ARM Cortex-M4 and is used in combination with the ChipWhisperer UFO board[5] and the open-source ChipWhisperer toolchain [48].

In our experiments, we send masked versions of key, nonce and plaintext to our target device. The software interface on the target device corresponds to the one defined in the call for protected software implementations of the NIST standardization process for lightweight cryptography[6]. Since our implementations do not require online randomness, we can use a simple software RNG on the target device to generate the necessary additional randomness for the initial sharing of the Ascon-p state without much performance impact. We then measure the runtime (cycles) of processing one block of plaintext (i.e. 8 bytes) without the overhead of initialization and finalization. The resulting numbers are presented in Table 3.

Compared to a plain (unmasked) implementation, the masked variants using 2 (3) shares have a runtime that is increased by a factor of about 6 (10). Even though the pure algorithmic overhead of our masking schemes is a lot lower than that, the main explanation for the observed runtime is the comparably small register file of Cortex-M4 (16×32-bit) and the resulting register spilling when computing on the shared state.

Since the Ascon AEAD mode allows the usage of so-called leveled implementations that provide protection against DPA-based key-recovery attacks using masking only during the initialization/finalization phases, we also present performance numbers for this case. To no surprise, our leveled implementation of Ascon using 2 or 3 shares can processes plaintext blocks with a similar performance to plain implementations. Nevertheless, with a bit more work, it should be possible to design a leveled implementation of Ascon that matches the throughput of the plain implementation.

[4] `https://rtfm.newae.com/Targets/UFO%20Targets/CW308T-STM32F/`
[5] `https://rtfm.newae.com/Targets/CW308%20UFO/`
[6] `https://cryptography.gmu.edu/athena/LWC/Call_for_Protected_Software_Implementations.pdf`

IBEX For our performance evaluation on the RISC-V IBEX core, we perform a cycle-accurate simulation of the IBEX netlist while executing one round of our masked software implementations of ASCON-p. The used assembly code and netlist is the same that was used for our formal verification efforts in Section 4. We then extrapolate the required cycles of processing one block of plaintext by multiplying the measured cycle count of one round by 6 (round parameter b) and dividing by 8 (block size in bytes). The results are presented in Table 3 and show a generally better performance than the ARM devices. This is mainly due to the fact that the register file of the IBEX core (32×32-bit) causes significantly reduced register spilling.

Implementation	`STM32F303`	IBEX
Plain	59	-
Leveled	89	-
2-shares	318	260*
3-shares	542	500*

*Estimated based on cycle counts of linear and non-linear layer.

Table 3: Performance of ASCON-128 for processing a single plaintext block on 32-bit microprocessors in cycles/byte ($X+0$ encrypt for long messages).

5.2 Practical Side-Channel Evaluation

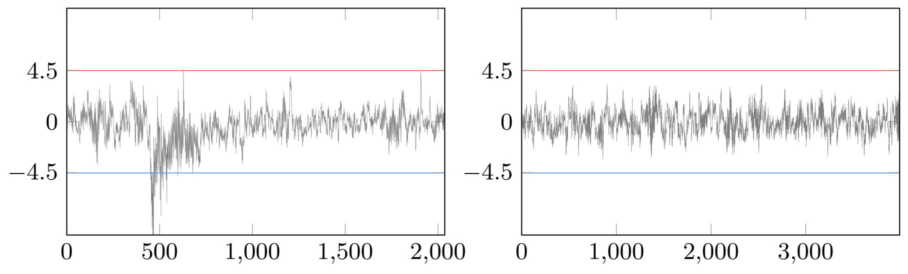
In our practical side-channel evaluation we perform standard test-vector leakage assessment (TVLA) following the guidelines of Goodwill [36], which is a standard method to measure information leakage of masked software implementations. The basic idea behind TVLA is to create two sets of power measurements, one corresponding to the processing of random inputs, and one corresponding to fixed inputs. Given such sets of measurements, one can compare their first and second-order statistical moments, i.e., mean and variance. The null-hypothesis is that both sets of measurements have equal means/variances, which is rejected with a confidence greater than 99.999% if the absolute t-score does not exceed the value 4.5. In this case, the sets of measurements cannot be reliably distinguished from each other and the masking countermeasure works as intended.

In our evaluation, we call masked versions of the ASCON-128 authenticated encryption procedure running on a `STM32F303` microprocessor using a fixed key, fixed or random nonces, and zero bytes of plaintext and associated data. The initial sharing of these inputs is generated using a proper source of randomness before transmitting them to the target device. The goal of our evaluation is to provide evidence that:

- `G1` Our 3-share implementations achieve practical second-order protection despite potential micro-architectural side-effects.

`G2` Our 3-share masking scheme remains secure over multiple rounds without any fresh randomness.

For this purpose, we perform multiple measurements covering one (four) rounds of Ascon-p during Ascon's initialization phase (cf. Figure 1) using a synchronized sampling frequency that is set to four (one) times the clock frequency. The restriction on the sampling rate for the four-round measurement is due to the fact that a bivariate analysis would otherwise become too computationally expensive. With our restrictions in place, a single measurement never contains much more than 4 000 samples. For the sake of comparison, we also provide measurements of 2-share implementations and implementations using a share rotation offset of zero which essentially deactivates our additional side-channel hardening technique. The power measurements themselves are recorded by a ChipWhisperer-Lite [48]. Given such sets of measurements, we evaluate their first/second-order statistical moments using the univariate and bivariate t-test functionality of the SCALib library[7]. The evaluation results of our 2 and 3-share Ascon implementations using 10M measurements are shown in Figure 2, Figure 3, and discussed in more detail in the following.



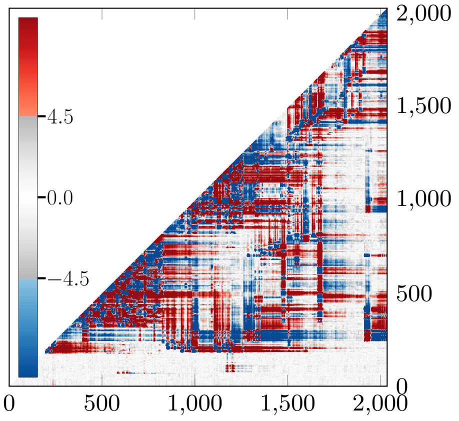
(a) shares:2, rounds:1, share-rotation:5. (b) shares:3, rounds:1, share-rotation:5.

Fig. 2: Univariate t-test of Ascon-p using 10M traces on the `STM32F303`.

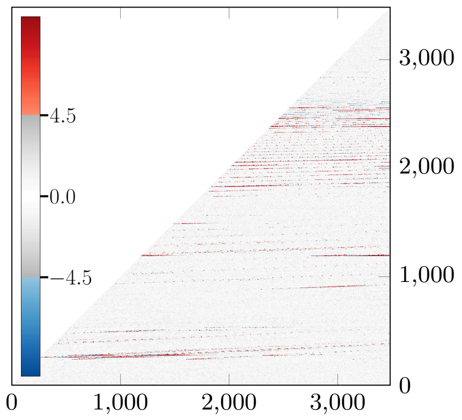
2 Shares Our 2-share implementation is based on the χ_{2S} construction from Section 3.1 and features share-rotations as an additional hardening technique (cf. Section 3.3). As can be seen in Figure 2a, our 2-share implementation does show some first-order leakage in the univariate t-test. While this should not happen in theory, many works in the past have pointed out that the practical security of masked software does depend on some assumptions that may not be satisfied on real processors due to microarchitectural side-effects such as transitions or glitches. Hence, without concrete knowledge of the microarchitecture of the target device, a certain reduction in practical protection order

[7] `https://github.com/simple-crypto/SCALib`

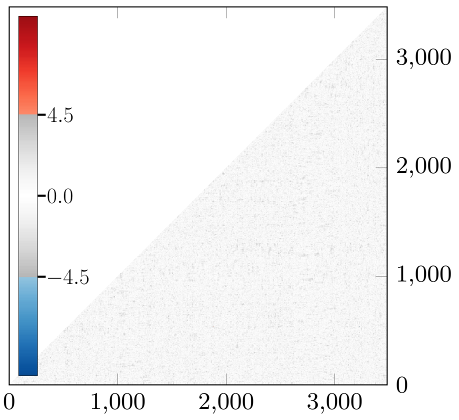
is not unexpected [49,37,43,59,35]. While hardening techniques such as share-rotations (cf. Section 3.3) can significantly reduce such unwanted side-effects, these measures were not sufficient to fully prevent first-order leakage in our measurements. As expected, if we take a look at the bivariate t-test result of our 2-share implementation in Figure 3a, we see a clear indication of second-order leakage after evaluating 10M traces.



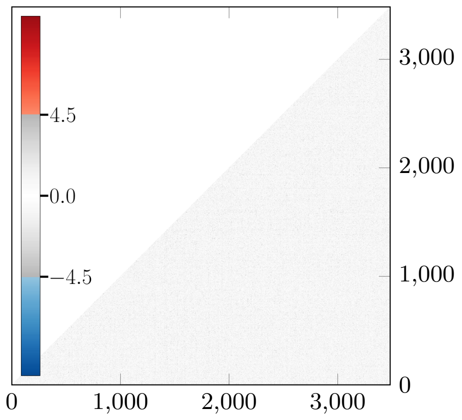
(a) shares:2, rounds:1, share-rotation:5.



(b) shares:3, rounds:1, share-rotation:0.



(c) shares:3, rounds:1, share-rotation:5.



(d) shares:3, rounds:4, share-rotation:5.

Fig. 3: Bivariate t-test of Ascon-p using 10M traces on the `STM32F303`.

3 Shares Our 3-share implementation is based on the $\chi_{3S^\star}$ construction from Section 3.3 that we have also formally verified for the correctness of masking in Section 4. Again, we use share-rotations as an additional hardening technique (cf. Section 3.3). Somewhat expected, as can be seen in Figure 2b, our 3-share implementation does not show any significant first-order leakage in our univariate t-test evaluation after 10M traces. In case of bivariate analysis, and corresponding to our evaluation goal `G1`, Figure 3b and Figure 3c show the practical difference of using a share-rotation hardening technique for our 3-share implementations. While a rotation offset of zero already leads to significant second-order leakage after evaluating only 10 000 traces, a non-zero offset results in no significant second-order leakage after evaluating 10M traces, thereby noticeably improving the practical side-channel security of our implementations. Generally, a certain security degradation of the masking due to transitions/glitches in the microarchitecture, or even more measurements, is also possible here. However the magnitude of these problems was not large enough to be of practical concern in our bivariate evaluation scenario where an attacker is forced to work with combinations of samples (and thus also their combined noise). Regarding our evaluation goal `G2`, Figure 3d shows a bivariate analysis covering four rounds of Ascon-p and gives no indication that the practical side-channel security of our implementation degrades over the course of four rounds, despite the fact the we do not use any fresh randomness during the computation.

6 Conclusion

We have presented efficient protected software implementations of the authenticated cipher Ascon targeting theoretical and practical security against second-order power analysis attacks. Our designs use a second-order extension of a previously presented first-order masking of the Keccak S-box based on Toffoli gates. This allows us to implement second-order masked software implementations of Ascon that do not require any online randomness and are hence especially suitable for the execution on low-end microprocessor devices. Our implementations also feature some implementation tricks that reduce the chance of unintended combinations of shares during the execution on microprocessors which helps them to preserve their theoretical protection against power analysis attacks in practice.

We benchmark our masked software implementations on 32-bit ARM and RISC-V microprocessors platforms and verified the practical and theoretical correctness of our masked implementations using TVLA on ARM microprocessors, as well as formal verification using Coco on the netlist of a RISC-V Ibex core. On both platforms, our second-order masked implementation of Ascon-128 reaches a throughput of about 550 cycles/byte or 90 cycles/byte if the leveled implementation technique is used.

While we do use the Ascon cipher as a discussion example, our techniques are also applicable to other lightweight symmetric ciphers, such as Keccak-like ciphers, or ciphers using 4-bit S-boxes that can be expressed as a sequence of Toffoli gates. We publish our software implementations together with generic software framework based on the ChipWhisperer toolchain that allows performance and side-channel evaluations of various masked cryptographic algorithms.

References

1. Albrecht, M.R., Rechberger, C., Schneider, T., Tiessen, T., Zohner, M.: Ciphers for MPC and FHE. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology – EUROCRYPT 2015. LNCS, vol. 9056, pp. 430–454. Springer (2015). https://doi.org/10.1007/978-3-662-46800-5_17
2. Balasch, J., Gierlichs, B., Grosso, V., Reparaz, O., Standaert, F.: On the cost of lazy engineering for masked software implementations. In: Smart Card Research and Advanced Applications - 13th International Conference, CARDIS 2014, Paris, France, November 5-7, 2014. Revised Selected Papers. Lecture Notes in Computer Science, vol. 8968, pp. 64–81. Springer (2014)
3. Bao, Z., Guo, J., Ling, S., Sasaki, Y.: Sok: Peigen - a platform for evaluation, implementation, and generation of s-boxes. IACR Cryptol. ePrint Arch. p. 209 (2019)
4. Bar-El, H., Choukri, H., Naccache, D., Tunstall, M., Whelan, C.: The sorcerer's apprentice guide to fault attacks. Proceedings of the IEEE **94**(2), 370–382 (2006)
5. Barthe, G., Belaïd, S., Cassiers, G., Fouque, P., Grégoire, B., Standaert, F.: maskverif: Automated verification of higher-order masking in presence of physical defaults. In: Computer Security - ESORICS 2019 - 24th European Symposium on Research in Computer Security, Luxembourg, September 23-27, 2019, Proceedings, Part I. Lecture Notes in Computer Science, vol. 11735, pp. 300–318. Springer (2019)
6. Barthe, G., Belaïd, S., Dupressoir, F., Fouque, P., Grégoire, B., Strub, P.: Verified proofs of higher-order masking. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9056, pp. 457–485. Springer (2015)
7. Barthe, G., Belaïd, S., Dupressoir, F., Fouque, P., Grégoire, B., Strub, P., Zucchini, R.: Strong non-interference and type-directed higher-order masking. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016. pp. 116–129. ACM (2016)
8. Bayrak, A.G., Regazzoni, F., Novo, D., Ienne, P.: Sleuth: Automated verification of software power analysis countermeasures. In: Bertoni, G., Coron, J. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2013 - 15th International Workshop, Santa Barbara, CA, USA, August 20-23, 2013. Proceedings. Lecture Notes in Computer Science, vol. 8086, pp. 293–310. Springer (2013)
9. Beckers, A., Wouters, L., Gierlichs, B., Preneel, B., Verbauwhede, I.: Provable secure software masking in the real-world. In: COSADE. Lecture Notes in Computer Science, vol. 13211, pp. 215–235. Springer (2022)
10. Belaïd, S., Benhamouda, F., Passelègue, A., Prouff, E., Thillard, A., Vergnaud, D.: Private multiplication over finite fields. In: Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part III. Lecture Notes in Computer Science, vol. 10403, pp. 397–426. Springer (2017)
11. Bellizia, D., Bronchain, O., Cassiers, G., Grosso, V., Guo, C., Momin, C., Pereira, O., Peters, T., Standaert, F.: Mode-level vs. implementation-level physical security in symmetric cryptography - A practical guide through the leakage-resistance jungle. In: CRYPTO (1). Lecture Notes in Computer Science, vol. 12170, pp. 369–400. Springer (2020)

12. Beyne, T., Dhooghe, S., Zhang, Z.: Cryptanalysis of masked ciphers: A not so random idea. In: ASIACRYPT (1). Lecture Notes in Computer Science, vol. 12491, pp. 817–850. Springer (2020)
13. Biham, E., Shamir, A.: Differential fault analysis of secret key cryptosystems. In: CRYPTO. Lecture Notes in Computer Science, vol. 1294, pp. 513–525. Springer (1997)
14. Bilgin, B., Nikova, S., Nikov, V., Rijmen, V., Tokareva, N.N., Vitkup, V.: Threshold implementations of small S-boxes. Cryptography and Communications **7**(1), 3–33 (2015). `https://doi.org/10.1007/s12095-014-0104-7`
15. Bloem, R., Groß, H., Iusupov, R., Könighofer, B., Mangard, S., Winter, J.: Formal verification of masked hardware implementations in the presence of glitches. In: Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II. Lecture Notes in Computer Science, vol. 10821, pp. 321–353. Springer (2018)
16. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: PRESENT: an ultra-lightweight block cipher. In: CHES. Lecture Notes in Computer Science, vol. 4727, pp. 450–466. Springer (2007)
17. Boneh, D., DeMillo, R.A., Lipton, R.J.: On the importance of checking cryptographic protocols for faults (extended abstract). In: EUROCRYPT. Lecture Notes in Computer Science, vol. 1233, pp. 37–51. Springer (1997)
18. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: CHES. Lecture Notes in Computer Science, vol. 2523, pp. 13–28. Springer (2002)
19. Coron, J., Giraud, C., Prouff, E., Renner, S., Rivain, M., Vadnala, P.K.: Conversion of security proofs from one leakage model to another: A new issue. In: Constructive Side-Channel Analysis and Secure Design - Third International Workshop, COSADE 2012, Darmstadt, Germany, May 3-4, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7275, pp. 69–81. Springer (2012)
20. Daemen, J.: Cipher and hash function design, strategies based on linear and differential cryptanalysis. Ph.D. thesis, KU Leuven (1995), `http://jda.noekeon.org/`
21. Daemen, J.: Changing of the guards: A simple and efficient method for achieving uniformity in threshold sharing. In: CHES. Lecture Notes in Computer Science, vol. 10529, pp. 137–153. Springer (2017)
22. Daemen, J., Dobraunig, C., Eichlseder, M., Groß, H., Mendel, F., Primas, R.: Protecting against statistical ineffective fault attacks. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2020**(3), 508–543 (2020)
23. Daemen, J., Hoffert, S., Van Assche, G., Van Keer, R.: The design of Xoodoo and Xoofff. IACR Transactions on Symmetric Cryptology **2018**(4), 1–38 (2018). `https://doi.org/10.13154/tosc.v2018.i4.1-38`
24. Daemen, J., Peeters, M., Van Assche, G., Rijmen, V.: Nessie proposal: the block cipher Noekeon. Nessie submission (2000), `http://gro.noekeon.org/`
25. De Cannière, C.: Analysis and design of symmetric encryption algorithms. Ph.D. thesis, KU Leuven (2007)
26. Dobraunig, C., Eichlseder, M., Korak, T., Mangard, S., Mendel, F., Primas, R.: SIFA: exploiting ineffective fault inductions on symmetric cryptography. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2018**(3), 547–572 (2018)
27. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Ascon v1.2. Submission to the NIST Lightweight Crypto Competition (2019), `https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/ascon-spec-round2.pdf`

28. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Ascon v1.2: Lightweight authenticated encryption and hashing. J. Cryptol. **34**(3), 33 (2021)
29. Eldib, H., Wang, C., Schaumont, P.: Formal verification of software countermeasures against side-channel attacks. ACM Trans. Softw. Eng. Methodol. **24**(2), 11:1–11:24 (2014)
30. Faust, S., Rabin, T., Reyzin, L., Tromer, E., Vaikuntanathan, V.: Protecting circuits from leakage: the computationally-bounded and noisy cases. In: Gilbert, H. (ed.) Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6110, pp. 135–156. Springer (2010)
31. Fuhr, T., Jaulmes, É., Lomné, V., Thillard, A.: Fault attacks on AES with faulty ciphertexts only. In: FDTC. pp. 108–118. IEEE Computer Society (2013)
32. Gaspoz, J., Dhooghe, S.: Threshold implementations in software: Micro-architectural leakages in algorithms. IACR Transactions on Cryptographic Hardware and Embedded Systems **2023**(2), 155–179 (Mar 2023). `https://doi.org/10.46586/tches.v2023.i2.155-179`
33. Gigerl, B., Hadzic, V., Primas, R., Mangard, S., Bloem, R.: Coco: Co-design and co-verification of masked software implementations on cpus. In: USENIX Security Symposium. pp. 1469–1468. USENIX Association (2021)
34. Gigerl, B., Primas, R., Mangard, S.: Secure and efficient software masking on superscalar pipelined processors. In: ASIACRYPT (2). Lecture Notes in Computer Science, vol. 13091, pp. 3–32. Springer (2021)
35. Gigerl, B., Primas, R., Mangard, S.: Formal verification of arithmetic masking in hardware and software. IACR Cryptol. ePrint Arch. p. 849 (2022)
36. Goodwill, G., Jun, B., Jaffe, J., Rohatgi, P.: A testing methodology for side-channel resistance validation. In: NIST Non-Invasive Attack Testing Workshop (2011)
37. de Groot, W., Papagiannopoulos, K., de la Piedra, A., Schneider, E., Batina, L.: Bitsliced masking and ARM: friends or foes? In: Lightweight Cryptography for Security and Privacy - 5th International Workshop, LightSec 2016, Aksaray, Turkey, September 21-22, 2016, Revised Selected Papers. Lecture Notes in Computer Science, vol. 10098, pp. 91–109. Springer (2016)
38. Groß, H., Mangard, S., Korak, T.: Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. In: Proceedings of the ACM Workshop on Theory of Implementation Security, TIS@CCS 2016 Vienna, Austria, October, 2016. p. 3. ACM (2016)
39. Groß, H., Mangard, S., Korak, T.: An efficient side-channel protected AES implementation with arbitrary protection order. In: CT-RSA. Lecture Notes in Computer Science, vol. 10159, pp. 95–112. Springer (2017)
40. Ishai, Y., Sahai, A., Wagner, D.A.: Private circuits: Securing hardware against probing attacks. In: Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2729, pp. 463–481. Springer (2003)
41. Knichel, D., Sasdrich, P., Moradi, A.: SILVER - statistical independence and leakage verification. IACR Cryptol. ePrint Arch. **2020**, 634 (2020)
42. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: CRYPTO. Lecture Notes in Computer Science, vol. 1666, pp. 388–397. Springer (1999)
43. Meyer, L.D., Mulder, E.D., Tunstall, M.: On the effect of the (micro)architecture on the development of side-channel resistant software. IACR Cryptol. ePrint Arch. **2020**, 1297 (2020)

44. Moss, A., Oswald, E., Page, D., Tunstall, M.: Compiler assisted masking. In: Prouff, E., Schaumont, P. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7428, pp. 58–75. Springer (2012)
45. Nikova, S., Rijmen, V., Schläffer, M.: Secure hardware implementation of nonlinear functions in the presence of glitches. J. Cryptol. **24**(2), 292–321 (2011)
46. NIST: Lightweight cryptography. `https://csrc.nist.gov/Projects/lightweight-cryptography` (2018)
47. NIST Lightweight Cryptography Team: Lightweight cryptography standardization process: NIST Selects Ascon (2023), `https://www.nist.gov/news-events/news/2023/02/lightweight-cryptography-standardization-process-nist-selects-ascon` (accessed 02/2023)
48. O'Flynn, C., Chen, Z.D.: Chipwhisperer: An open-source platform for hardware embedded security research. In: COSADE. Lecture Notes in Computer Science, vol. 8622, pp. 243–260. Springer (2014)
49. Papagiannopoulos, K., Veshchikov, N.: Mind the gap: Towards secure 1st-order masking in software. In: Constructive Side-Channel Analysis and Secure Design - 8th International Workshop, COSADE 2017, Paris, France, April 13-14, 2017, Revised Selected Papers. Lecture Notes in Computer Science, vol. 10348, pp. 282–297. Springer (2017)
50. Pereira, O., Standaert, F., Vivek, S.: Leakage-resilient authentication and encryption from symmetric cryptographic primitives. In: CCS. pp. 96–108. ACM (2015)
51. Quisquater, J., Samyde, D.: Electromagnetic analysis (EMA): measures and counter-measures for smart cards. In: E-smart. Lecture Notes in Computer Science, vol. 2140, pp. 200–210. Springer (2001)
52. Renauld, M., Standaert, F., Veyrat-Charvillon, N., Kamel, D., Flandre, D.: A formal study of power variability issues and side-channel attacks for nanoscale devices. In: Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6632, pp. 109–128. Springer (2011)
53. Reparaz, O., Bilgin, B., Nikova, S., Gierlichs, B., Verbauwhede, I.: Consolidating masking schemes. In: CRYPTO (1). Lecture Notes in Computer Science, vol. 9215, pp. 764–783. Springer (2015)
54. Reparaz, O., Bilgin, B., Nikova, S., Gierlichs, B., Verbauwhede, I.: Consolidating masking schemes. In: Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9215, pp. 764–783. Springer (2015)
55. Rivain, M., Prouff, E.: Provably secure higher-order masking of AES. In: CHES. Lecture Notes in Computer Science, vol. 6225, pp. 413–427. Springer (2010)
56. Rivain, M., Prouff, E.: Provably secure higher-order masking of AES. In: Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6225, pp. 413–427. Springer (2010)
57. Shahmirzadi, A.R., Moradi, A.: Re-consolidating first-order masking schemes nullifying fresh randomness. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2021**(1), 305–342 (2021)

58. Shahmirzadi, A.R., Moradi, A.: Second-order SCA security with almost no fresh randomness. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2021**(3), 708–755 (2021)
59. Shelton, M.A., Samwel, N., Batina, L., Regazzoni, F., Wagner, M., Yarom, Y.: Rosita: Towards automatic elimination of power-analysis leakage in ciphers. CoRR **abs/1912.05183** (2019)
60. Shende, V.V., Prasad, A.K., Markov, I.L., Hayes, J.P.: Synthesis of reversible logic circuits. IEEE Transactions on CAD of Integrated Circuits and Systems **22**(6), 710–722 (2003). `https://doi.org/10.1109/TCAD.2003.811448`
61. Sugawara, T.: 3-share threshold implementation of AES s-box without fresh randomness. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2019**(1), 123–145 (2019)
62. Zhang, J., Gao, P., Song, F., Wang, C.: Scinfer: Refinement-based verification of software countermeasures against side-channel attacks. In: Chockler, H., Weissenbacher, G. (eds.) Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part II. Lecture Notes in Computer Science, vol. 10982, pp. 157–177. Springer (2018)

Appendix: Ascon-*p*

The following description of the Ascon-p permutation is adapted from the Ascon specification [27]. The permutation iteratively applies an SPN-based round transformation p that in turn consists of three steps p_C, p_S, p_L and differ only in the number of rounds:

$$p = p_L \circ p_S \circ p_C \, .$$

For the description and application of the round transformations, the 320-bit state S is split into five 64-bit registers words x_i, $S = x_0 \parallel x_1 \parallel x_2 \parallel x_3 \parallel x_4$.

Addition of Constants

The constant addition step p_C adds a round constant c_r to register word x_2 of the state S in round i. Both indices r and i start from zero and we use $r = i$ for p^a and $r = i + a - b$ for p^b (see Table 4):

$$x_2 \leftarrow x_2 \oplus c_r \, .$$

Substitution Layer

The substitution layer p_S updates the state S with 64 parallel applications of the 5-bit S-box $\mathcal{S}(x)$ defined in Figure 4a to each bit-slice of the five registers $x_0 \ldots x_4$. It is typically implemented in bitsliced form with operations performed on the 64-bit words.

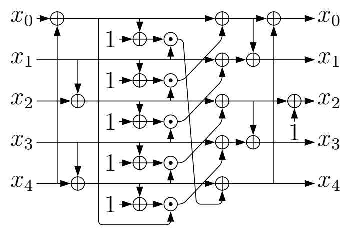
Table 4: The round constants c_r used in each round i of ASCON.

p^{12}	p^8	p^6	Constant c_r	p^{12}	p^8	p^6	Constant c_r
0			`00000000000000f0`	6	2	0	`0000000000000096`
1			`00000000000000e1`	7	3	1	`0000000000000087`
2			`00000000000000d2`	8	4	2	`0000000000000078`
3			`00000000000000c3`	9	5	3	`0000000000000069`
4	0		`00000000000000b4`	10	6	4	`000000000000005a`
5	1		`00000000000000a5`	11	7	5	`000000000000004b`

Linear Diffusion Layer

The linear diffusion layer p_L provides diffusion within each 64-bit register word x_i. It applies a linear function $\Sigma_i(x_i)$ defined in Figure 4b to each word x_i:

$$x_i \leftarrow \Sigma_i(x_i), \quad 0 \le i \le 4.$$



(a) ASCON's 5-bit Sbox $\mathcal{S}(x)$

$$x_0 \leftarrow \Sigma_0(x_0) = x_0 \oplus (x_0 \ggg 19) \oplus (x_0 \ggg 28)$$
$$x_1 \leftarrow \Sigma_1(x_1) = x_1 \oplus (x_1 \ggg 61) \oplus (x_1 \ggg 39)$$
$$x_2 \leftarrow \Sigma_2(x_2) = x_2 \oplus (x_2 \ggg 1) \oplus (x_2 \ggg 6)$$
$$x_3 \leftarrow \Sigma_3(x_3) = x_3 \oplus (x_3 \ggg 10) \oplus (x_3 \ggg 17)$$
$$x_4 \leftarrow \Sigma_4(x_4) = x_4 \oplus (x_4 \ggg 7) \oplus (x_4 \ggg 41)$$

(b) ASCON's linear layer with 64-bit functions $\Sigma_i(x_i)$

Fig. 4: ASCON's substitution layer and linear diffusion layer