Higher-Order Side-Channel Protected Implementations of KECCAK

Hannes Gross, David Schaffenrath, Stefan Mangard

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

Abstract. The efficient protection of security critical devices against side-channel analysis attacks is a fundamental need in the age of Internet of Things and ubiquitous computing. In this work, we introduce a configurable hardware design of KECCAK (SHA-3) which can be tailored to fulfill the needs of a wide range of different applications. Our KECCAK design is therefore equipped with generic side-channel protection capabilities. The design can thus be synthesized for any desired protection level by just changing one design parameter.
Regardless of its generic appearance, the introduced KECCAK design yields the smallest (15.7 kGE) first-order protected KECCAK implementation published to this date. Furthermore, it is to the best of our knowledge the first higher-order side-channel resistant implementation of KECCAK. In total, we state results for four different KECCAK variants up to the ninth protection order.

Keywords: Keccak, SHA-3, masking, domain-oriented masking, threshold implementations, DPA

1 Introduction

The resistance to side-channel analysis (SCA) attacks is a very important requirement for many security-critical devices. If no appropriate countermeasures are implemented, an attacker can easily reconstruct sensitive data from physical observations like the power consumption [16] or the electromagnetic emanation [20].

Masking has proven to be a very effective countermeasure against SCA and was extensively researched over the last fifteen years [11, 15, 23]. Many of the earlier schemes were shown to be insecure for hardware implementations, because signal propagation effects (glitches) were not considered in their construction. The first scheme that achieved first-order security under consideration of glitches is the so-called threshold implementation scheme (TI) by Nikova *et al.* [17]. This work has been extended to achieve higher-order security in the univariate setting by Bilgin *et al.* [4]. Reparaz *et al.* [21] extended this approach to achieve security in the multivariate setting, and lowered the sharing complexity by requiring only $d+1$ input shares for a d^{th}-order secure circuit. Gross *et al.* [12] demonstrated that the randomness cost for $d+1$ TI can be lowered from $(d+1)^2$ to $d(d+1)/2$ by using a domain-oriented masking (DOM) approach.

However, fitting masked hardware implementations to the requirements of specific applications is a difficult task that often requires extensive redesigning effort when a design needs to be adapted to meet the given constraints.

Our contribution. In this work, we introduce a very versatile KECCAK design, that can be configured to fit many different use cases and requirements, from high-throughput to small-area applications. In addition, our KECCAK design can be fully customized in terms of SCA protection to meet the desired level of security. Therefore, we built upon the work of Gross *et al.* [13] that introduced a generic DOM AES implementation which allows synthesis for arbitrary protection orders.

To increase the efficiency of the protected implementation, we investigate different optimizations for our design and evaluate their impact. As a result, our smallest first-order protected KECCAK variant requires only 15.7 kGE of chip area.

It is thus the smallest and least randomness demanding SCA protected KECCAK implementation that has been reported to this date. We demonstrate the genericity of our design by stating results of four variants of our KECCAK design up to the ninth protection order. The source-code of our design is published online to make our results reproducible and better accessible for future research [22]. We round off our contribution with a side-channel evaluation of our protected designs based on a Welch's t-test [10].

This paper is organized as follows: In Section 2 we discuss the basic structure of KECCAK . In Section 3 we give a brief overview of the used DOM scheme. Section 4 introduces optimizations that help to reduce the chip-area and randomness overhead of our implementations. In Section 5 we describe our implementation and compare it to existing ones in Section 6. In Section 7 we evaluate the side-channel resistance of our implemented designs. Section 8 concludes the paper.

2 KECCAK

KECCAK is a family of sponge functions, from which several instantiations have been standardized by NIST as SHA-3, SHAKE and KECCAK-*p* in [18], as a result from the SHA-3 hash competition. The SHA-3 specification describes four different instantiations of the KECCAK-*f* [1600] hash function. The KECCAK family [18] consists of seven permutations KECCAK-*f* [b], for $b \in \{25, 50, 100, 200, 400, 800, 1600\}$, where b denotes the width of the permutation. These permutations are organized in a sponge construction, which also allows to express the KECCAK permutation in terms of rate r and capacity c as KECCAK [r,c], for $b = r + c$. The rate in the sponge construction corresponds to the block size and can be a multiple of a lane size, while its capacity determines the security level as $c/2$.

Although the seven permutations of KECCAK-*f* [b] have different permutation widths, their underlying round function is always the same. A full round of KECCAK consists of the five steps θ, ρ, π, χ and ι, which operate on the three-dimensional state in this order. A full permutation is defined as the repeated application of these five steps.

– θ is a linear diffusion step. It calculates the parity of each column in a slice and adds it to a neighboring column in the same and the next higher slice.
– ρ and π are also linear diffusion steps, most often implemented directly by wiring in a hardware implementation.
– χ is a degree-2 non-linear mapping. It operates on each row of the state independently and is implemented as $x_i \leftarrow x_i + (x_{i+1} + 1)x_{i+2}$.
– ι is a simple addition of a round constant to a lane.

3 Domain-Oriented Masking

The basic idea of masking is to make the power consumption of a hardware implementation independent from the processed data in order to mitigate side-channel analysis attacks. Therefore, the sensitive data is first split into a number of so-called shares which we denote by capital letters with the name of the shared variable in the index. Accordingly, a variable a is split into to a bundle of shares named A_a, B_a, C_a, et cetera, which are picked uniformly at random. There exist an additive relation over $GF(2^n)$ between these shares so that $a = A_a + B_a + C_a + \ldots$ is always valid. The resistance against side-channel analysis is then achieved by keeping these shares separated throughout the whole circuit.

For the domain-oriented masking approach (DOM) [12] this independence is achieved by splitting the circuit into $d + 1$ subcircuits called share domains (where d is the so-called protection order). Each share of a circuit variable is then associated with one specific domain indicated by the leading capital letter of the share. The share A_a, *e.g.* , is associated with domain A, the share B_a accordingly with domain B. By keeping these domains independent from the other domains, the data dependency of the overall power consumption is shifted to a higher statistical moment. As it was shown by Chari *et al.* [6], by shifting the leakage into a higher statistical moment, a successful attack requires exponentially more leakage traces in relation to d which significantly hardens power-analysis attacks. While a correct implementation of functions that are linear over $GF(2^n)$ can be trivially implemented for each domain separately by keeping the shares in their respective domains, non-linear functions require the secure exchange of information between the domains. To still fulfill the independence requirement at all points in time, the introduction of fresh randomness is sometimes required.

First-Order Protection. Figure 1 shows a first-order protected example for a non-linear GF multiplier (calculating $q = ab$ in shared form) which corresponds in the case of $GF(2)$ to a masked one-bit AND gate. This masked multiplier is a fundamental building block in DOM and for our KECCAK implementation. The depicted DOM multiplier uses two domains A and B, and is first-order secure for two independent input sharings of the variables a and b. The sharings are provided by the shares A_a and B_a, and A_b and B_b, respectively. At all times the equation $a = A_a + B_a$ and $b = A_b + B_b$ must be ensured.

The equations implemented by the multiplier are shown in Equation 1 where the parentheses indicate registers. The fresh random share Z (respectively A_c and B_c, for the optimization explained in Section 4) is required to keep independence for the multiplication terms that source different domains (red parts of the circuit). By adding Z to the cross-domain signals, these signals are made independent again from all other shares. The connected register ensures that the Z share is added before the integration of the cross-domain signal is performed and so prevents glitches.

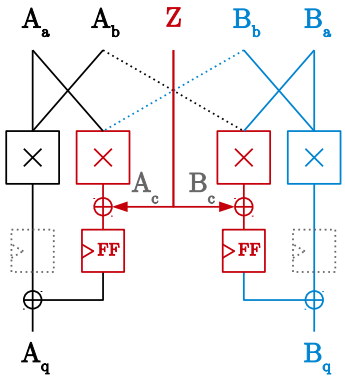

Fig. 1. First-order multiplier calculating $q = ab$, and with randomness optimization for $q = ab + c$ calculation (grey)

$$\begin{aligned} A_q &= A_a A_b + (A_a B_b + Z) \\ B_q &= B_a B_b + (B_a A_b + Z) \end{aligned} \tag{1}$$

Differences to MPC. Please note, even though the similarities between masking and multi-party computation (MPC) are evident, their assumptions at some point are different. For example, in MPC the assumption is that an attacker has only access to some domains (up to a certain threshold) but inside these domains, the attacker has access to all variables. In this sense the multiplier in Figure 1 would be automatically flawed because all shares of b are used for the calculation of A_q and B_q. In the probing model [15] which is commonly used to formalize the security of masking, the security is argued over the number of probing needles an attacker requires to gain information on an unshared security-critical variable. Probing a certain gate or wire thus only reveals the information going in or out of the gate or the information on the wire. The masked multiplier is secure because at no point during the computation an attacker can get information on all shares of a, b, or q by using d (= 1 for first-order masking) probes.

Higher-Order Protection. The construction of the secure GF multiplier can be generalized to arbitrary protection orders [12] as shown in Equation 2. Therefore, the domains are first enumerated by $A \rightarrow 0$, $B \rightarrow 1$, $C \rightarrow 2$, ... using the indexes i or j, respectively. The equation then states the domain function Q_i where $0 \leq i \leq d$ and $Q_0 \rightarrow A_q$, $Q_1 \rightarrow B_q$, $Q_2 \rightarrow C_q$, et cetera. The variable $t_{i,j}$ refers to multiplication terms in which i is used to address shares of the variable a and j as index of shares of the variable b. The term $t_{0,0}$, for example, is the multiplication term $A_a A_b$, and $t_{1,2}$ is the term $B_a C_b$.

$$\mathrm{Q_i} = t_{i,i} + \sum_{j>i}^{d} (t_{i,j} + Z_{(i+j*(j-1)/2)}) + \sum_{j<i}^{d} (t_{i,j} + Z_{(j+i*(i-1)/2)}) \tag{2}$$

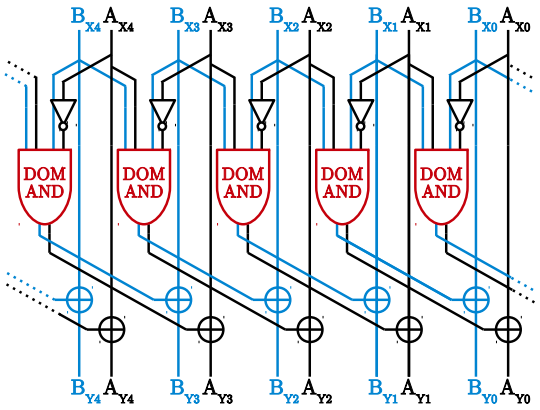

Fig. 2. First-order protected S-box of KECCAK with the DOM multiplier from Figure 1

The first term $t_{i,i}$ is a so-called inner-domain term which is associated with the i^{th} domain. Inner-domain terms do not require a fresh random Z share in DOM because, similar to linear functions, here the shares are kept in the same domain. For the $t_{i,j}$ cross-domain terms used inside the sums, a fresh random Z bit is required before the integration into the targeted domain can be performed. Splitting of the sums is required for the correct and secure distribution of the Z shares. In this way it is ensured that the term $t_{i,j}$ is masked by the same fresh random Z share as the mirrored term $t_{j,i}$ which is used in another domain.

This generic multiplier construction allows to design generically masked hardware implementations that are secure for any given parameter d. The area requirements of the generic multiplier increases quadratically with the protection order d and the randomness costs are given as $d(d+1)/2$. This secure multiplier design is the basis for the construction of our KECCAK designs.

4 Optimizations

In order to make our protected KECCAK implementation more efficient, we introduce different optimizations to decrease the overhead in terms of area, delay, and required randomness.

Randomness Optimization. DOM was intended for the efficient and generic higher-order protection of hardware designs. This genericity, however, leads to an unnecessary overhead in terms of fresh randomness for first-order protection of some S-box constructions. In particular, this affects S-boxes of the form $ab + c$ which is the case *e.g.* in the S-box of KECCAK , Present [5], Noekeon [9], and LowMC [1] .

The 5-bit KECCAK S-box is given as $x_i+(x_{i+1}+1)x_{i+2}$ which is of the form $ab + c$. In a first order DOM protected KECCAK S-box (see Figure 2), the straightforward implementation requires 5 bit of randomness for each S-box to calculate $x_{i+1}x_{i+2}$. However, the shares of x_i are independent from the ones of x_{i+1} and x_{i+2} and thus the shares of x_i fulfill the same property as a fresh random Z share. Instead of adding Z to the cross-domain terms in both domains, we instead add the shares associated with x_i as A_c and B_c to these cross-domain terms in their respective domain (see Figure 1).

Saving randomness by reusing unrelated state bits has already been reported for first-order threshold implementations by Bilgin *et al.* [3] and more recently by Daemen *et al.* [7]. The difference is that Daemen's changing-of-the-guards approach performs an explicit resharing at the end of the S-box function for two out of five S-box bits, which requires additional XOR gates, while instead we perform this implicitly.

While the probing security of the construction in Figure 1 is trivially given under the assumption that the sharings of the input bits (a, b, and c) are independent, the security argumentation gets more difficult when the rest of the KECCAK transformations and the full S-box implementation is considered. Indeed the uniformity of the state bits and therefore their suitability for masking other operations degenerates over the rounds of KECCAK as stated by Daemen. As this effect is considered to be minimal by Daemen [8] and therefore unpractical to exploit, we still consider it as a valid option in our design. However, we note that, formally speaking, this optimization leads to a flaw in the probing assumption. Using this optimization requires careful investigation of the degeneration effect for the targeted design or usage of the changing-of-the-guards method from Daemen [7]. Thus we made the usage of this optimization optional for our first-order KECCAK instantiation.

Throughput and Area Optimization. The DOM multiplier always introduces a delay cycle through the resharing of the cross-domain terms. To make this calculation more efficient, another register could be added in the inner-domain paths of the multiplier to generate a pipeline stage (grey dotted registers in Figure 1). However, this has a negative influence on the required chip area if many S-box instances are used in parallel. We circumvent the additional register by clocking the cross-domain flip-flops on the negative clock edge, which effectively means doubling the clock frequency for the S-box. We investigate the effectiveness of this approach in Section 6 and investigate its influence on the maximum throughput. Furthermore, we note that this optimization is only a meaningful option if the critical path in the S-box is relatively small compared to the other circuit parts, or the overall clock frequency of the chip is somewhat already constrained. Otherwise this optimization might have a negative influence on the layout of the design to meet the clock constraints.

5 Implementation

Our generic KECCAK implementation allows to be customized for a variety of requirements for different security critical applications. In the following, all possible configurations of the KECCAK design are explained, and the main variants which are considered in Section 6 are introduced.

A high-level architectural view of the design is shown in Figure 3. It is important to note, that each mapping is either connected to the next one directly *or* to the sponge state. The configuration of the connection is done at synthesis time. Hence the connection between the steps, respectively between a mapping and the sponge state, is done by simple wiring. Everything from a fully parallel implementation, in which all five steps are done in one clock cycle, to a fully iterative one, in which the output of each step is written back into the state, can be instantiated. For example when omitting the gray connections shown in Figure 3,

the SERIAL-AREA configuration, which is described later in this Section, is obtained.

Iterative Application of Functions. The design allows to apply the individual round transformation steps in an iterative way, which takes either multiple cycles, but can also be performed in a single clock cycle. In case a step is handled iteratively it will act on a specified number of state slices (in powers of two) in parallel. This means that the state can be thought of as a number of FIFOs. The FIFO's output form the input of the iteratively applied step, and the output of the step gets either piped into the input of the following step (which would in turn need to be iterative), or back to the FIFO. The specified number of parallel slices is used for all iterative steps (except for an iterative ρ/π step which is explained further down). This simplifies the design by allowing to chain the slice-based iterative χ, ι, θ and absorption steps together.

The iterative version of the steps looks as follows.

- In θ, one output slice depends on the parity of a previously processed slice's columns. This can simply be handled by storing the parity of the highest processed slice of the previous cycle. This works for every but the first slice, which can only be finished once we look at the last slice. Thus the first slice is a special case and is finished together with the last one.
- An iterative ρ step means that each lane gets rotated until the desired offset is reached. This does imply some control overhead, but allows to save most of the multiplexers which are needed when $\pi \circ \rho$ is performed in one cycle, which requires the full state to be written at once.
- The π mapping is then applied together with χ, which works out nicely, since both are slice based functions and π is implemented by simple wiring.
- The iterative ι step is simply done by only adding the relevant part of the round constant. In our implementation ι is always done concurrently with the S-box function χ.

It is also possible to choose the lane length freely, which means, that any KECCAK-f [b] variant can be instantiated.

Absorption. A concrete instantiation of our design can perform the absorption either in a lane-based or a slice-based fashion. In case of a slice-based absorption, the absorbed slice(s) can be directly fed into an iterative θ step, which saves cycles that otherwise would be wasted solely for absorption. In the case of a lane-based absorption such optimizations are not possible. However, lane-based absorption often fits much more naturally with how data is processed and sent over buses, hence possibly saving area or increasing overall performance, depending on the concrete system. To avoid having to include additional buffers in case of systems with bus widths unequal to the lane length, it is possible to adjust the number of bits absorbed in a single cycle (in powers of two). This means, that it is possible to absorb more than one lane at once, or only a fraction of a lane, as long as the word to absorb is a power of two, depending on the configuration.

Concrete instantiations. Since the number configurations which are possible with this approach is huge, we focus on three corner cases.

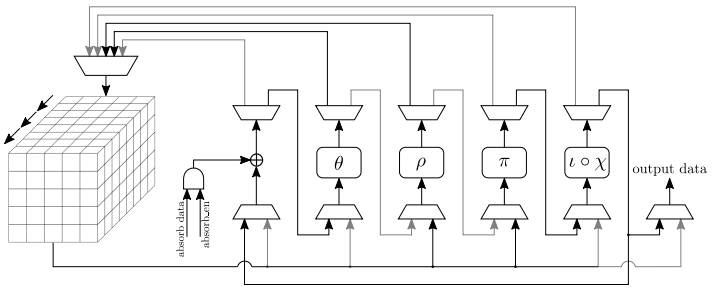

Fig. 3. Simplified architecture of our implementation.

- SERIAL-TP All steps except ρ and π are performed iteratively. The absorption is done in a slice-based manner, in parallel with the θ step in the first round. ρ and π are done in a separate step. The χ and ι steps are chained together with the absorption XORs and the θ step. Thus the processing of a block takes $r(\frac{W}{SP}+1)$ cycles, where W is the lane length, SP are the number of parallel processed slices and r the number of rounds. This implementation is similar to the one described by Bilgin *et al.* [3] but performs $\pi \circ \rho$ in a dedicated cycle instead of concurrently with the last cycle of θ.
- SERIAL-AREA This variant is similar to SERIAL-TP but every step is done iteratively (including ρ and π). In the iterative ρ step, each lane gets rotated until a counter exceeds the rotation offset of that lane, hence this step now takes W cycles to complete. This saves most of the multiplexers which are needed when $\pi \circ \rho$ is performed in one cycle, because not the full state needs to be updated at once. This simple modification yields the smallest register-based implementation of KECCAK to date. As a trade-off, throughput is decreased, compared to SERIAL-TP, since ρ now takes W cycles to complete.
- PARALLEL A fully parallel implementation in which we try to achieve the highest possible throughput. The unprotected instantiation performs $\iota \circ \chi \circ \pi \circ \rho \circ \theta$ in one cycle. The DOM-masked instantiations require an additional flip-flop stage before the S-boxes, otherwise glitches in the θ step would lead to a violation of the probing security during the application of the χ step [2]. Hence for one round of KECCAK $\pi \circ \rho \circ \theta$ is performed in one cycle and $\iota \circ \chi$ in the next. Such a configuration yields the highest throughput but on the other hand requires $5 \cdot 2^l$ 5-bit S-boxes.

S-Box Variants. For the any of the previously described configurations, we further instantiated two different variants, that differ in the implementation of the S-Box in the χ step.

- *Pipelined* DOM variant. In this variant we use additional inner-domain flip-flops together with the cross-domain flip-flops as a pipeline stage for the multiplier in Figure 1.

[2] Please note, in the original DSD [14] version the PARALLEL instantiations didn't have this register stage. We'd like to thank Victor Arribas, Begül Bilgin, Svetla Nikova and Vincent Rijmen for pointing-out this flaw. The results were updated accordingly.

– *Double clocked* DOM variant. Here we try to keep the area overhead of DOM minimal by saving the inner-domain flip-flops and clocking the cross-domain flip-flops on the negative clock edge as described in Section 4.

Both S-box variants use the optional optimization to reduce randomness in the first-order case, as described in Section 4. For the SERIAL configurations the overhead of the inner-domain flip-flops is negligible when only one slice is processed in parallel. Thus only the pipelined S-box variant is shown in the results for these instantiations.

Area Estimation. For the protected implementations, the linear parts (except for the inverts) need to be replicated for each additional domain. Hence the linear θ, ρ, π mappings, as well as the state itself are expected to scale linearly with the number of shares $(d+1)$. For the non-linear χ step, a more detailed look at the S-Box is required. An unprotected S-Box consists of 5 AND, 5 NOT and 5 XOR gates as can be obtained when only looking at domain A (black parts) in Figure 2 and replacing the DOM AND instances by normal AND gates. An estimation for the scaling of the χ step can be given by looking at the generic construction of the DOM AND gate in Equation 2, and replicating the XOR gates for each share, as illustrated in Figure 2 for two shares. Generally speaking, for $d+1$ shares, the combinatorial part of the S-Box consists of $5(d+1)^2$ AND gates, 5 NOT gates and $5(d+1)+5(d+1)^2$ XOR gates. Simplifying this by looking only at the DOM AND gate ($(d+1)^2$ ANDs and XORs), we can estimate the combinatorial part to increase with a factor $(d+1)^2$. The number of flip-flops in the χ step depends on whether the implementation is pipelined (uses the inner-domain flip-flops) or not. The pipelined variant requires $(d+1)^2$ flip-flops, the variant without inner-domain flip-flops requires $d(d+1)$.

6 Results

The synthesis results are obtained with the configurations described in Section 5. We apply the configurations to the KECCAK [1088,512] permutation, since it is a SHA3 standard, and allows comparisons with other publications. All values have been obtained by synthesizing the design with Cadence RTL Compiler version 8.1 XL. We used the FSC0H_D and FSD0A_A libraries from FARADAY for the 130 nm and 90 nm designs respectively. The numbers given in plots correspond to designs synthesized with the 130 nm library.

A detailed look at the synthesis results, up to second protection order, is given in Table 1. The KECCAK team itself were the first to provide a first-order protected KECCAK threshold implementation with three shares [2]. The implementation was later on improved by Bilgin *et al.* [3] resulting in the smallest register-based[1] protected and unprotected KECCAK designs reported up to now.

Area Requirements. The SERIAL configurations in Table 1 show the resulting numbers when processing a single slice per cycle. This allows to directly compare our serial designs with the one of Bilgin *et al.* [3], which also processes one slice per cycle. Their unprotected design has a size of just 10.6 kGE for their serial implementation while our unprotected variants use 11 kGE in case of SERIAL-TP, and 9.2 kGE in the SERIAL-AREA implementation for the cost of a doubled cycle count. This makes our SERIAL-AREA configuration the smallest register-based KECCAK implementation reported so far.

When looking at the SERIAL configurations' increase in size between unprotected (11.0/9.2 kGE), first order (22.3/18.7 kGE) and second order (34.6/28.8 kGE) protected SERIAL designs in Table 1, it can be seen that all linear parts grow linearly with the protection order as expected. This is also illustrated in Figure 4, which shows that the area requirement increases almost linearly with the protection order for the SERIAL designs. The only non-linear part of the design is the χ step, which only operates on one slice in the SERIAL configurations as shown in Table 1, and thus the χ transformation has only a marginal influence on the overall size.

For the PARALLEL configurations, the linear parts of the design grow linearly with the protection order as well. The non-linear χ step now operates on the full state, hence $64 \cdot 5$ 5-bit S-Boxes are required, making it the main contributor to the overall area.

As discussed in Section 5, the area of the DOM protected χ step increases non-linearly with the number of shares. This can best be observed in the area increase from the unprotected PARALLEL to the protected PARALLEL configurations. The unprotected χ step (6.4kGE) consists solely of combinational logic, a 1600-bit state of small flip-flops is around 8.8kGE (see SERIAL-AREA in Table 1). Hence, as was discussed in Section 5, a rough area estimation for the χ step of the parallel KECCAK implementation with $d+1$ shares would be $(d+1)^2 \cdot (6.4\text{kGE} + 8.8\text{kGE})$ for the pipelined and $(d+1)^2 \cdot 6.4\text{kGE} + d(d+1) \cdot 8.8\text{kGE}$ for the double-clocked variant. In case of a first-order protected implementation this corresponds to 60.8kGE, respectively 43.2kGE, which is close to the actually achieved results (57.6kGE, resp. 44.2kGE).

Compared to the existing implementations our designs have a lower area overhead for the same protection order, while achieving similar throughput, in all configurations. The main reason for this difference is that related work uses the $dt+1$ TI approach which requires at least three input shares for first-order protection while our $d+1$ share implementations require only two shares in the first-order case.

Throughput Considerations In case higher throughput is desired it is also possible to increase the number of slices that are processed in parallel as mentioned in Section 5. Figure 5 shows how the area and maximum frequency develop when doing so for the pipelined SERIAL-TP configuration. Note, that while the area and maximum frequency for SERIAL-AREA would look similar, the throughput gain would be lower, due to ρ always taking 64 cycles.

The throughput of the SERIAL-TP configuration doubles if the number of parallel processed slices doubles. This of course also doubles the number of needed S-boxes, thus the area increase with higher protection order becomes less linear. Compared to the first order protected $dt+1$ TI in Table 1 that needs 32.6 kGE , the DOM protected counterpart uses just 22.3 kGE.

The highest throughput is achieved with the PARALLEL configuration, which needs the full 1600 S-boxes. The area require-

[1] The smallest design this far is achieved by the usage of RAM macros and needs significantly more cycles per block [19].

Prot. order	Design	UMC 0.13μm Area (kGE) θ	χ	State/Other	$\sum$	Freq. (MHz)	UMC 90nm Area (kGE) θ	χ	State/Other	$\sum$	Freq. (MHz)	Cycles	Rand.
None	PARALLEL	8.6	6.4	16.2	31.2	919.1	7.4	6.0	14.0	27.4	1,287.0	24	-
	SERIAL-TP	0.3	0.1	10.6	11.0	866.6	0.2	0.1	9.6	9.9	861.3	1624	-
	SERIAL-AREA	0.3	0.1	8.8	9.2	861.3	0.2	0.1	7.4	7.7	900.9	3136	-
1st order	PARALLEL double clocked [2]	17.2	44.2	38.9	100.5	803.9	15.0	38.4	32.2	85.7	891.3	48	-
	PARALLEL pipelined [2]	17.2	57.6	36.8	111.8	837.5	15.0	50.4	32.2	97.7	846.7	72	-
	SERIAL-TP pipelined	0.6	0.9	20.8	22.3	812.3	0.4	0.8	18.9	20.1	864.3	1648	-
	SERIAL-AREA pipelined	0.6	0.9	17.1	18.7	856.2	0.4	0.4	14.5	15.7	850.3	3160	-
2nd order	PARALLEL double clocked [2]	26.0	139.9	60.1	226.0	811.7	22.5	114.0	51.1	188.1	897.7	48	4800/cycle
	PARALLEL pipelined [2]	25.8	157.2	55.1	238.4	840.3	22.5	138.0	48.2	208.9	848.9	72	4800/cycle
	SERIAL-TP pipelined	1.0	2.5	31.1	34.6	844.6	0.6	2.2	28.1	30.9	845.3	1648	75/cycle
	SERIAL-AREA pipelined	0.9	2.5	25.4	28.8	852.5	0.6	2.2	21.4	24.2	898.5	3160	75/cycle
Related Work													
None	Parallel [3]	8.6	6.4	15.6	30.6	855	-	-	-	-	-	24	-
	Serial [3]	0.1	0.1	10.4	10.6	752	-	-	-	-	-	1600	-
1st order	Parallel-3sh [3]	25.7	52.8	56.7	135.2	746	-	-	-	-	-	25	4/round
	Parallel-4sh [3]	34.2	61.6	61.8	157.6	735	-	-	-	-	-	24	-
	Serial-3sh [3]	0.4	0.8	31.4	32.6	820	-	-	-	-	-	1625	4/round
	Serial-4sh [3]	0.5	0.9	41	42.4	775	-	-	-	-	-	1600	-

Table 1. Synthesis results

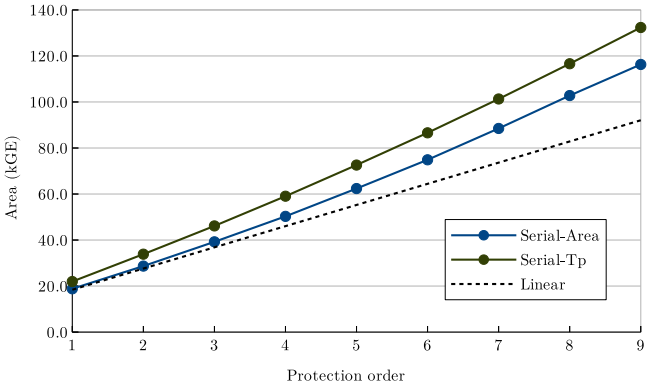

Fig. 4. Area requirement for increasing number of share domains. SERIAL with 1 slice processed in parallel with pipelined S-box

ment can be lowered by implementing the double clocking of the S-box as described in Section 4. As shown in Table 1, the area of the double clocked S-box variant (100.5 kGE resp. 226.0 kGE) is noteworthy smaller, than of the pipelined S-box variant (111.8 kGE resp. 238.4 kGE).

7 Side-Channel Evaluation

To test our KECCAK implementations' resistance to side-channel analysis attacks, a Welch's t-test is used which is a standard test for masked hardware implementations (see Goodwill *et al.* [10] for details). The t-test basically compares two distributions and outputs a statistical value t that serves as basis for the decision whether or not the mean values of the two sets of measurements are equal. Therefore, two different trace sets A and B are collected, one set of traces for randomly picked inputs and another one for traces for constant input values. For both sets the sharing of the inputs is always generated randomly but the value of the unshared inputs differ. Finally, the t-value for the two sets is calculated according to Equation 3 where the means of the respective set are indicated by X, the variances are given by S^2, and the set sizes by N. The null-hypothesis is that the mean values of both sets are equal, which means that random inputs cannot be distinguished from constant inputs. This equals the requirement for first-order protection. For second order t-tests, a normalized product combining preprocessing step is applied. The null-hypothesis is rejected if the t-value exceeds the ± 4.5 border.

$$t = \frac{X_A - X_B}{\sqrt{\frac{S_A^2}{N_A} + \frac{S_B^2}{N_B}}} \tag{3}$$

Measurement Setup. Originally, t-tests were designed to test the leakage of a protected (hardware) implementation on its targeted platform (*e.g.* a specific FPGA or ASIC design). Since we do not target any specific platform, and in order to deal with the complexity of higher-order multivariate t-test assessment for multiple first and higher-order designs, we decided on using leakage traces from a post-synthesis netlist simulation to generate toggle-count traces for individual signals. Compared to other practical evaluations, an evaluation based on simulated traces has some clear benefits but also a few drawbacks. While measurements performed on real devices are highly dependent on environmental and operating conditions, and thus noisy, post-synthesis simulation traces are noise free. If there is any exploitable leakage available, the statistical evidence for this is thus found much faster, with less leakage traces, and with higher confidence. Additionally it is possible to focus on the critical parts of an implementation and the noise caused by irrelevant parts of the circuit may be ignored. This approach even allows to analyze where a leakage originates from by performing the described t-test on (combinations of) individual circuit signals.

Signal timing differences introduced by gates can already be observed in the post-synthesis netlist simulation and thus many possible glitches are covered by this approach. Flaws like in the masked AND gate of the ISW [15] scheme, for example, are detected by this approach. Also possible degeneration effects in our first-order optimized KECCAK implementation would be recognized with significantly less traces than for an FPGA or ASIC

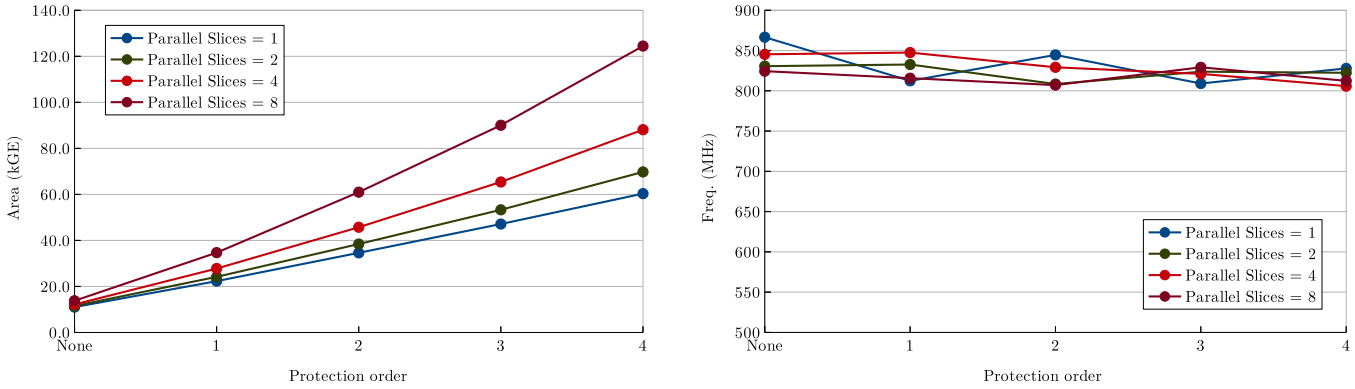

Fig. 5. SERIAL-TP: Area and frequency over the number of share domains for different number of parallel processed slices

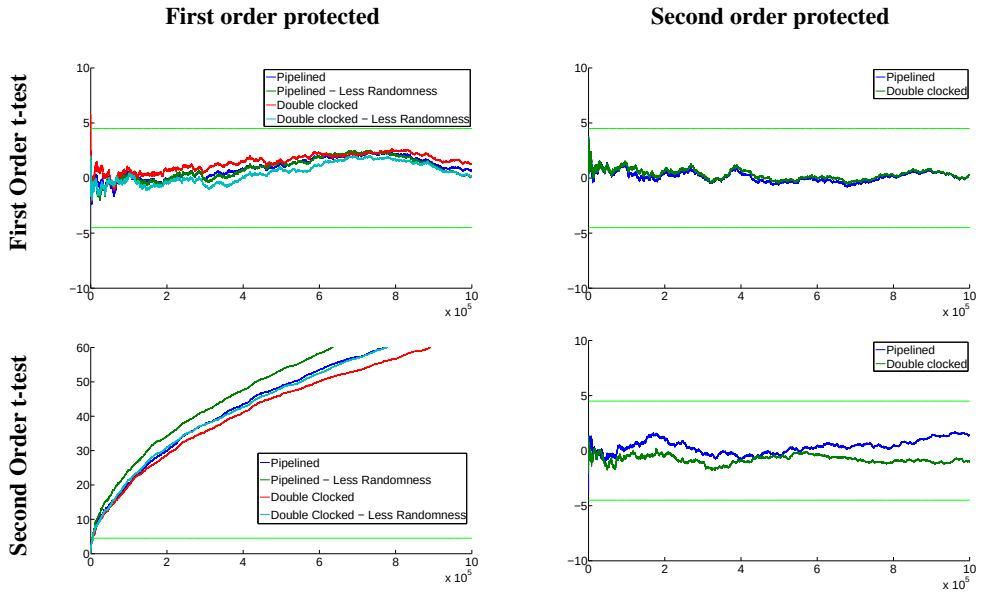

Fig. 6. SCA evaluation of the χ step for different attack and protection orders

evaluation, since this effect does not depend on signal timings. We note that because of the simple delay model used for the post-synthesis traces, it is possible that a practical evaluation catches some more glitches then our targeted approach. However, we emphasize that a t-test either way can never replace a formal verification as it would be required to ensure leakage freeness up to a certain order for all possible signal timings and under all environmental conditions. Since, to the best of our knowledge, there exist no such formal tools for hardware implementations up to now, we keep the formal verification as future work and note that our evaluation is only meant to strengthen the reader's trust in the carefulness we put into designing the KECCAK implementations.

Evaluation Target. The χ step of our designs is the most vulnerable circuit part since it is the only non-linear step, thus the only that requires the combination of different share domains. In order to get reproducible results, we use a seeded PRNG, using different seeds for the random and constant sets. The mean and variance are calculated incrementally, to show the evolution of the t-value for an increasing numbers of traces up to 1 million noise-free traces.

Since the χ step is considered to be the source of any potential leakage in the design, we restrict the simulation to the χ step exclusively. This has the advantage, that not only simulation is considerably faster, but we also don't introduce unnecessary noise (meaning toggling of uninteresting signals) from the rest of the system, hence significant t-values should be observable with a greatly reduced amount of traces.

In order to verify that no implementation errors were made, which cause leakage somewhere else than the χ step, we additionally simulated a first-order protected KECCAK-f [25] with and without the reduced-randomness tweak from Section 4. We applied constant-shared and random inputs for absorption to this KECCAK-f [25], as described previously, and let it process the data for twelve rounds, as is specified for this state size, before reiterating these steps. No significant t-values could be observed when performing a first-order t-test on the traces obtained this way.

Evaluation results. The results of performing first and second-order t-tests on different protected χ variants with and without the optimizations described in Section 4 (double-clocking and randomness optimization) are shown in Figure 6. The significance

levels, which are shown in light green, show the t-values at ± 4.5 which corresponds to a confidence interval of 99.9995% for rejecting the null-hypothesis.

The first-order t-test for the first-order and second-order protected KECCAK variants do not indicate any leakage as targeted by our implementation. The second-order t-test for the first-order protected implementations, on the other hand, exceed the confidence border even after a few hundred traces. This shows the correctness of our evaluation setup and the sensitivity of the t-test in this noise-free setting. Please note, that the number of traces required to break a masked implementation grows exponentially with the protection order [6] and also highly depends on the signal to noise ration of the attacked leakage signal. For a practical DPA attack, an attacker would thus require significantly more measurements to exploit this second-order leakage than indicated by this t-test results on noise-free leakage traces. As expected, the second-order leakage disappears for the second-order protected variant.

Furthermore, it can be seen that our optimizations do not affect the protection level in a negative way.

8 Conclusions

In this work, we introduced a generic side-channel protected KECCAK design which allows to be customized to fit many different requirements from low area to high throughput. The design thus suits a wide range of security critical applications. Without touching the hardware design itself but only its configuration, all different kinds of KECCAK and SHA-3 variants can be synthesized for any desired order of side-channel protection (see [22] for the Verilog sources).

We furthermore investigated and evaluated different possibilities to lower the overhead introduced through masking in terms of area, throughput, and required randomness. As an outcome, our design yields the smallest and least randomness demanding protected KECCAK implementations at any protection level reported to this date.

Acknowledgements.

We'd like to thank Victor Arribas, Begül Bilgin, Svetla Nikova, and Vincent Rijmen for pinpointing a flaw in the parallel variant of our original DSD paper [14]. This work has been supported by the Austrian Research Promotion Agency (FFG) under grant number 845589 (SCALAS). The HECTOR project has received funding from the European Unions Horizon 2020 research and innovation programme under grant agreement No 644052. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 681402).

References

1. M. R. Albrecht, C. Rechberger, T. Schneider, T. Tiessen, and M. Zohner. Ciphers for MPC and FHE. In *EUROCRYPT 2015*, 2015.
2. G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, and R. Van Keer. Keccak implementation overview. *URL: http://keccak.neokeon.org/Keccak-implementation-3.2.pdf*, 2012.
3. B. Bilgin, J. Daemen, V. Nikov, S. Nikova, V. Rijmen, and G. Van Assche. Efficient and First-Order DPA Resistant Implementations of Keccak. In *CARDIS 2014*, LNCS. Springer International Publishing, 2014.
4. B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen. Higher-Order Threshold Implementations. In P. Sarkar and T. Iwata, editors, *Advances in Cryptology ASIACRYPT 2014*, volume 8874 of *LNCS*. 2014.
5. A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT: an ultra-lightweight block cipher. In *CHES 2007*, 2007.
6. S. Chari, C. Jutla, J. Rao, and P. Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In M. Wiener, editor, *CRYPTO 99*, volume 1666 of *LNCS*. 1999.
7. J. Daemen. Changing of the Guards: a simple and efficient method for achieving uniformity in threshold sharing. Cryptology ePrint Archive, Report 2016/1061, 2016. `http://eprint.iacr.org/2016/1061`.
8. J. Daemen. On Non-uniformity in Threshold Sharings. In *Proceedings of the 2016 ACM Workshop on Theory of Implementation Security*, TIS '16, pages 41–41, New York, NY, USA, 2016. ACM.
9. J. Daemen, M. Peeters, G. Van Assche, and V. Rijmen. Nessie proposal: Noekeon. 2000.
10. G. Goodwill, B. Jun, J. Jaffe, and P. Rohatgi. A Testing Methodology for Side-Channel Resistance Validation. In *NIST Non-Invasive Attack Testing Workshop*, 2011.
11. L. Goubin and J. Patarin. DES and Differential Power Analysis The Duplication Method. In *Cryptographic Hardware and Embedded Systems*, volume 1717 of *LNCS*. 1999.
12. H. Gross, S. Mangard, and T. Korak. Domain-Oriented Masking: Compact Masked Hardware Implementations with Arbitrary Protection Order. In *Proceedings of the 2016 ACM Workshop on Theory of Implementation Security*, TIS '16, pages 3–3, New York, NY, USA, 2016. ACM.
13. H. Gross, S. Mangard, and T. Korak. An Efficient Side-Channel Protected AES Implementation with Arbitrary Protection Order. In *Topics in Cryptology - CT-RSA 2017 - The Cryptographers' Track at the RSA Conference 2017, San Francisco, CA, USA, February 14-17, 2017, Proceedings*, pages 95–112, 2017.
14. H. Groß, D. Schaffenrath, and S. Mangard. Higher-order side-channel protected implementations of KECCAK. In *DSD*, pages 205–212. IEEE Computer Society, 2017.
15. Y. Ishai, A. Sahai, and D. Wagner. Private Circuits: Securing Hardware against Probing Attacks. In D. Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, volume 2729 of *LNCS*. 2003.
16. P. C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '99, London, UK, 1999. Springer-Verlag.
17. S. Nikova, C. Rechberger, and V. Rijmen. Threshold Implementations Against Side-Channel Attacks and Glitches. In P. Ning, S. Qing, and N. Li, editors, *Information and Communications Security*, volume 4307 of *LNCS*. 2006.
18. NIST. *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*, 2015.

19. P. Pessl and M. Hutter. Pushing the limits of sha-3 hardware implementations to fit on rfid. In *Cryptographic Hardware and Embedded Systems – CHES 2013, 15th International Workshop, Santa Barbara, CA, USA, August 20-23*, volume 8086. Springer, 2013.
20. J.-J. Quisquater and D. Samyde. ElectroMagnetic Analysis (EMA): Measures and Counter-measures for Smart Cards. In *Smart Card Programming and Security*, volume 2140 of *LNCS*. 2001.
21. O. Reparaz, B. Bilgin, S. Nikova, B. Gierlichs, and I. Verbauwhede. Consolidating Masking Schemes. In *CRYPTO 2015*, 2015.
22. D. Schaffenrath. DOM Protected Hardware Implementation of Keccak. `https://github.com/hgrosz/keccak_dom`, 2017.
23. E. Trichina. Combinational logic design for AES subbyte transformation on masked data. *IACR Cryptology ePrint Archive*, 2003, 2003.