Trade-offs in protecting Keccak against combined side-channel and fault attacks

Antoon Purnal, Victor Arribas, and Lauren De Meyer

KU Leuven, imec - COSIC, Belgium
{firstname.lastname}@esat.kuleuven.be

Abstract. When deployed in a potentially hostile environment, security-critical devices are susceptible to physical attacks. Consequently, cryptographic implementations need to be protected against side-channel analysis, fault attacks and attacks that combine both approaches. CAPA (CRYPTO 2018) is an algorithm-level combined countermeasure, based on MPC, with provable security in a strong attacker model. A key challenge for combined countermeasures, and CAPA in particular, is the implementation cost. In this work, we use CAPA to obtain the first hardware implementations of Keccak (SHA-3) with resistance against combined side-channel and fault attacks. We systematically explore the speed-area trade-off and show that CAPA, in spite of its algorithmic overhead, can be very fast or reasonably small. In fact, for the standardized Keccak-f[1600] instance, our low-latency version is nearly twice as fast as the previous implementations that only consider side-channel security, at the cost of area and randomness consumption. For all four presented designs, the protection level for side-channel and fault attacks can be scaled separately and to arbitrary order. To evaluate the physical security, we assess the side-channel leakage of a representative second-order secure implementation on FPGA. In addition, we experimentally validate the claimed fault detection probability.

Keywords: Side-channel analysis · Fault attacks · Masking · Combined countermeasure · Keccak · SHA-3 · CAPA

1 Introduction

Computing devices implement cryptographic algorithms. Traditionally, these devices are assumed to operate out of the attacker's reach. In practice, however, this condition is not upheld. Next to cryptanalytic attacks, the adversary can target the implementation of the algorithm directly. On the one hand, an adversary can employ *side-channel analysis* (SCA), which exploits the unintended leakage of sensitive information through one or more side-channels. Measurable physical channels include timing [28], power consumption [29] and electromagnetic emanation [16]. On the other hand, an adversary can mount devastating attacks by actively *injecting faults* in the cryptographic computations [12].

To thwart SCA attacks, masking [11,13,18,20,24,32,35,37] is a provably secure and scalable countermeasure. By randomizing the intermediate values processed

by the cryptographic algorithm, masking decouples the side-channel information from the actual sensitive values.

The lion's share of countermeasures against fault attacks perform some redundant computations that allow the *detection* of faults. Proposed solutions include duplication of the computations in space or time [3], the use of error-detecting codes [5,26,30] and recomputing with permuted operands [22,34]. While such approaches are intuitively sound, they suffer from two fundamental problems. To begin with, if the redundancy is predictable, the attacker can evade detection by introducing well-crafted faults. Moreover, the detection mechanism itself constitutes an interesting point of attack [27,39]. Conceptually different from detection, *infection* [17,40] avoids the vulnerable check-before-output procedure. Instead, injected faults perturb the computation in such a way that the cryptographic output reveals no information about the implementation's secrets.

An attacker capable of separate side-channel and fault attacks, can also jointly exploit both attack vectors. Hence, cryptographic implementations also need to be protected against *combined attacks*. The combined countermeasures Private Circuits II [14,23] and ParTI [38] are constructed by combining a masking scheme with fault-detecting redundancy. As a result, the fundamental problems of fault detection apply equally well to these schemes. The Capa [36] and M&M [15] countermeasure methodologies avoid the latter problem by employing information-theoretic (i.e. perfectly unpredictable) MAC tags. Capa draws inspiration from advances in the field of secure multi-party computation (MPC), resulting in provable security against combined physical attacks in a strong adversarial model. M&M is much cheaper to implement than Capa, at the expense of a weaker attacker model. It additionally addresses the fundamental problem of fault checking by using the aforementioned infection strategy.

A key challenge for combined countermeasures is the implementation cost. We contribute to the evaluation of combined countermeasures by investigating the hardware trade-offs that govern the widely used Keccak permutations [6] when protecting against combined physical attacks. We instantiate Capa because it is the most resource-intensive methodology, but the implementation strategies and conclusions carry over to other (combined) countermeasures for which multiplications dominate the implementation cost. By extension, this work also covers the authenticated encryption ciphers Ketje [8] and Keyak [9].

Our contribution. In this work, we present the first implementations of Keccak (SHA-3) with resistance to combined physical attacks, where previous works [1,7,10,21] have considered only side-channel analysis. We systematically explore the speed-area trade-off in hardware, yielding a suite of protected implementations. We show that, in spite of the extremely strong adversarial model, Capa can be very fast *or* reasonably compact. In particular, our low-latency implementation is almost twice as fast as all existing protected Keccak-f[1600] implementations. As a bonus, we discover a generic implementation optimization of the Capa preprocessing stage. We illustrate and experimentally validate the overhead cost of the countermeasure as a function of the Keccak permutation width and the side-channel and fault security parameters.

2 Preliminaries

2.1 KECCAK

Best known for their standardization as SHA-3, KECCAK [6] is a family of sponge functions, based on the KECCAK-$f[b]$ permutations. These permutations manipulate a state of b elements in $GF(2)$ (bits) for $b \in \{25, 50, 100, 200, 400, 800, 1600\}$ and consist of the iterative application of a round function R. Specifically, each of the seven instances of KECCAK-$f[b]$ has a fixed number of rounds $n_r = 12+2l$, where $l = \log_2(\frac{b}{25})$. The round function R, in turn, is defined by the consecutive application of five step mappings: $R = \iota \circ \chi \circ \pi \circ \rho \circ \theta$.

The effect of the step mappings is best explained by considering the state as a three-dimensional array $S(x, y, z)$ of dimensions $5 \times 5 \times w$, where $w = \frac{b}{25}$. This paper employs the established naming convention as introduced in [6]. In particular, we adopt the nomenclature of *planes, rows, lanes, slices and columns* to denote specific parts of the state.

The nonlinear part of the KECCAK-$f[b]$ permutation is confined to the χ step mapping, which is an S-box operating on 5-bit rows. Its algebraic degree is two, which is an attractive property in the context of masked implementations. The other step mappings are linear. For each column of the state, the θ mapping adds the parity of two neighbouring columns. The ι step mapping adds a round constant to one of the lanes. Finally, π reorganizes the lanes in the state and ρ shifts the bits within one lane.

2.2 CAPA: a combined countermeasure against physical attacks

CAPA [36] is an algorithm-level countermeasure that achieves resistance against attacks that simultaneously exploit side-channel leakage and fault injection. As such, it claims security in the *tile-probe-and-fault* adversarial model [36]. We consider a computing architecture that has been partitioned in d tiles, resulting in side-channel security up to order $d-1$. Let $\mathcal{T}_i$ denote one such tile and $\mathcal{T}$ the set of all tiles such that $\mathcal{T} = \bigcup_{i=0}^{d-1} \mathcal{T}_i$. The secure evaluation of an arbitrary arithmetic circuit occurs in two distinct stages. The *evaluation* stage comprises the actual cryptographic computations. The security of this stage depends on the presence of auxiliary random values, generated in the *preprocessing* stage.

The intermediate values of the cryptographic computation are referred to as sensitive variables x, y, z. The preprocessing stage generates auxiliary values a, b, c. Every sensitive or auxiliary value $x \in \mathbb{F}_q$ is shared as $\boldsymbol{x} = (x_0, x_1, \ldots, x_{d-1})$ where each tile $\mathcal{T}_i$ holds one share x_i and $\sum x_i = x$. The same sharing applies to every auxiliary value $a \in \mathbb{F}_q$. To detect faults injected in the evaluation stage, a MAC key $\alpha \in \mathbb{F}_q$, drawn uniformly at random, authenticates every sensitive or auxiliary value x with a multiplicative tag $\tau^x = \alpha \cdot x$, shared between the tiles as $\boldsymbol{\tau^x} = (\tau_0^x, \tau_1^x, \ldots, \tau_{d-1}^x)$. Note that α authenticates the secret value itself, not the shares. To protect α from being observed by the attacker, it is also shared between the tiles as $(\alpha_0, \alpha_1, \ldots, \alpha_{d-1})$. As α is secret, an attacker that alters a

sensitive value is generally unable to forge a valid tag. The MAC key α changes for every new execution of the cryptographic algorithm.

To obtain a scalable security level against fault attacks, CAPA considers m independent MAC keys $\alpha[j]$, such that each sensitive value x is accompanied by m tags $\tau^x[j]$, for $j = 0, 1, \ldots, m-1$. Because KECCAK operates in $\mathbb{F}_q = GF(2)$, the CAPA fault detection probability in this work is $1 - 2^{-m}$.

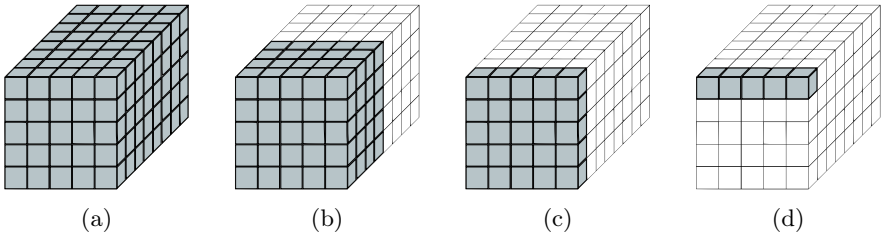
Evaluation stage. Next to establishing a shared representation $\langle \boldsymbol{x} \rangle = (\boldsymbol{x}, \boldsymbol{\tau^x})$ of every input value x of the arithmetic circuit, CAPA defines a computing procedure that yields a shared representation of the output. Linear operations, such as the parity additions in θ or the addition of the round constant in the ι step mapping, are easily evaluated by isolated computations in every tile. Nonlinear operations, like field multiplications, are more difficult to implement securely. In the CAPA methodology, multiplications require communication between the tiles and the consumption of an auxiliary triple $\langle \boldsymbol{a} \rangle, \langle \boldsymbol{b} \rangle, \langle \boldsymbol{c} \rangle$ that satisfies $c = a \cdot b$.

To be secure in the presence of glitches, all communication between the tiles has to be synchronized by registers. Referring to [36] for the details, this implies that a multiplication operation has a two-cycle latency. However, multiplications can be organised in a pipeline, reducing the impact on the throughput of the implementation. To detect faults injected in the evaluation stage, every multiplication also features a check of the MAC tag. In an optimized implementation, this MAC check completes one cycle later than its corresponding multiplication.

Preprocessing stage. Every multiplication instance in the evaluation stage incurs a corresponding preprocessing entity that produces the necessary auxiliary triple. In what follows, such an entity is denoted by a triple *factory*. The generation of an auxiliary triple goes as follows. First, the factory draws $\boldsymbol{a} = (a_0, a_1, ..., a_{d-1})$ and $\boldsymbol{b} = (b_0, b_1, ..., b_{d-1})$ uniformly at random from $\mathbb{F}_q^d$. To this end, every tile $\mathcal{T}_i$ randomly generates their share a_i, b_i. Next, the tiles securely compute a shared representation $\boldsymbol{c} = (c_0, c_1, ..., c_{d-1})$ of $c = a \cdot b$ by multiplying $\boldsymbol{a}$ and $\boldsymbol{b}$ with a *passively secure shared multiplier* [11,19,33,35]. The choice of multiplier is free, as long as the partition in tiles can be superimposed. In this work, we instantiate the DOM multiplier [19] because of its low randomness consumption. Shared multiplications of resp. $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$ with the MAC key $\boldsymbol{\alpha}$ in turn yield the tags $\boldsymbol{\tau_a}$, $\boldsymbol{\tau_b}$ and $\boldsymbol{\tau_c}$. Note that each new triple thus requires $(1 + 3m)$ shared multiplications. To detect faults in the preprocessing and hence verify that the Beaver triples are genuine, the factory sacrifices another triple satisfying the same relation. This procedure is explained in more detail in [36].

3 Protected implementations of Keccak

Due to the substantial area and randomness cost of nonlinear operations in the CAPA methodology, the selection of the number of S-boxes is a crucial design decision. In this work, we explore the *speed-area trade-off* by presenting four secure designs of KECCAK-$f[b]$: BLAZE, FAST, FUR and KIT.



(a) (b) (c) (d)

Fig. 1: Simultaneous processing of χ for the different designs.
(a) BLAZE. (b) FAST. (c) FUR. (d) KIT

The starting point for each design is the number of χ operations computed in parallel. Fig. 1 marks the bits of the state that are treated simultaneously in the χ mapping. The design choices for the other step mappings follow from the speed-area characteristics implied by the number of S-boxes.

3.1 Evaluation stage

BLAZE. The BLAZE design targets high throughput. Fig. 2 (left) depicts a high-level overview of this round-based architecture. The delay elements in χ, unavoidable to be secure in the presence of glitches, are used as pipelining registers. Combinational logic implements the other step mappings. In the integrated $\pi \circ \rho \circ \theta$ stage, π and ρ are simple wirings.

The first cycle initiates the χ pipeline and hence only computes $\pi \circ \rho \circ \theta$. The subsequent cycles compute $\pi \circ \rho \circ \theta \circ \iota \circ \chi$ and the permutation finishes in the final cycle with $\iota \circ \chi$. Before the result can be shown at the output, the MAC tag check pertaining to the final computation cycle needs to be performed.

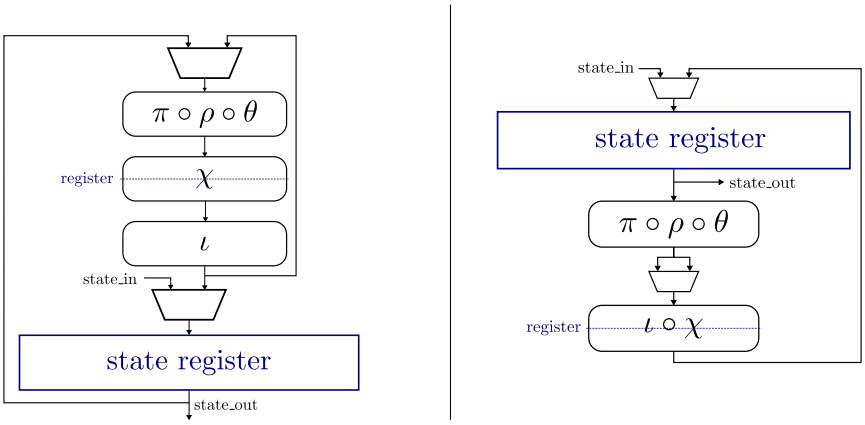


Fig. 2: High-level architectures for BLAZE (left) and FAST (right)

Fast. The baseline of the Fast design is a χ mapping that treats half of the state at a time. Fig. 2 (right) presents the high-level architecture for Fast. The $\iota \circ \chi$ stage partitions the state in two equal parts by treating the first $w/2$ slices in the first cycle and the remaining $w/2$ slices in the second cycle. The ι mapping follows the pace dictated by χ and adds the round constant in two parts. Since the inter-slice shifts in ρ do not match with the slice-based partition for χ, the $\pi \circ \rho \circ \theta$ stage must consider the entire state at once.

The computation of one round takes three cycles. In the first cycle, the first part of the state is loaded in the $\iota \circ \chi$ pipeline. During the second cycle, the now completed first part is written in the state register while the remaining part of the state enters the $\iota \circ \chi$ pipeline. In the third cycle, the now completed second part of the state also ends up in the state register. As with the Blaze design, one extra cycle is needed to verify the tags of the last $\iota \circ \chi$.

Fur. As shown in Fig. 1, the baseline for the Fur design is a *slice-based* treatment of χ. Slice-based designs naturally lead to smaller implementations owing to the reuse of functional units. Moreover, a great deal of multiplexers are saved because only one slice during χ is written at once, as opposed to the entire state. The latter insight is key, given that storing the state dominates the implementation size as we move towards smaller designs. Although the architecture is not exactly the same, we acknowledge that a slice-based paradigm for Keccak has been reported previously [25].

Schematically represented in Fig. 3, the slice-based $\iota \circ \chi$ stage takes one slice at a time, corresponding to one bit in every lane. At the same time, the slices are shifted and the result is written in the newly vacant position resulting from the shift. Because of the shift, the next slice is now in place to be processed. After repeating this process for all the slices, the results end up in the correct location. The $\pi \circ \rho \circ \theta$ mapping considers the whole state at once and its result is written to the state register to allow χ to be performed in a slice-based way. Due to its slice-based nature, χ takes $w + 1$ cycles, where the additional cycle stems from starting up the pipeline. One cycle suffices for $\pi \circ \rho \circ \theta$ and the ι mapping is computed concurrently with χ. In total, Fur takes $w + 2$ cycles per round, where one cycle for the MAC check must be added after the final round.

Kit. The Kit design performs every step mapping in an iterated fashion and considers the minimal number of χ modules, i.e. *one*. In particular, Kit employs the slice-based paradigm for $\pi \circ \theta$ and ρ and even a *row-based* paradigm for χ. Row-based processing can be seen as an even more area-efficient extension of slice-based processing, resulting in the smallest design of this paper.

The Kit architecture comprises three distinct stages: $\pi \circ \theta$, ρ and $\iota \circ \chi$. Note that this interchanges the order of ρ and π. The $\pi \circ \theta$ stage is slice-based and implemented as in [10]. When processing one slice, its column parities are precomputed for the next slice. The first slice features a special treatment as it requires the parities of the columns of the last slice. As a result, it is processed

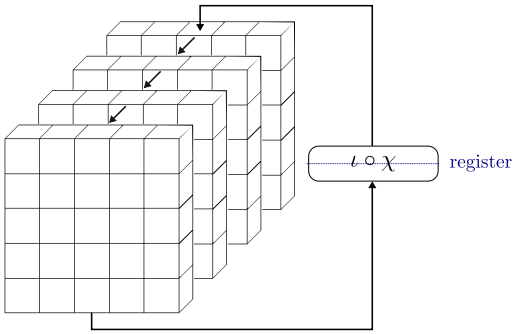


Fig. 3: Slice-based processing for the $\iota \circ \chi$ stage in FUR

together with the last slice. The inter-slice diffusion in the ρ stage is also implemented iteratively. In particular, the lanes are shifted circularly until they reach the configuration dictated by ρ. It should be mentioned that a slice-based ρ mapping has been reported previously [21].

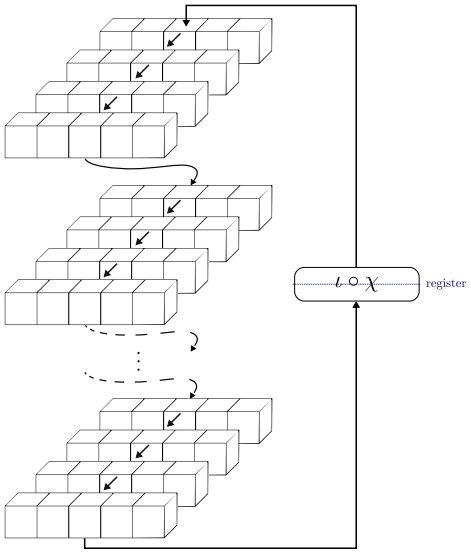


Fig. 4: Row-based processing for the $\iota \circ \chi$ stage in KIT

Fig. 4 clarifies the row-based paradigm. The $\iota \circ \chi$ stage takes one row and shifts the remaining rows in the plane to fill the newly vacant position. In turn, this leaves a vacancy at the last row, which is filled by the first row of the next plane. Continuation of this reasoning results in a vacancy in the last row of the final plane, where the $\iota \circ \chi$ output is written. This process is repeated for every

Table 1: Summary of the designs, generic in the permutation width b. Recall that $w = b/25$ and that n_r denotes the number of rounds.

Design	# S-boxes (χ)	# Factories	Cycle count
Blaze	$b/5$	b	$n_r + 2$
Fast	$b/10$	$b/2$	$3 \cdot n_r + 2$
Fur	5	25	$(w+1) \cdot n_r + 1$
Kit	1	5	$(7w+1) \cdot n_r + 1$

row. Care should be taken that the round constant is only added to the relevant lane. The row-based χ stage takes $5 \cdot w + 1$ cycles to treat the entire state. The other stages are slice-based and hence require w cycles each. In total, each round takes $(7w + 1)$ cycles.

Pushing the area limits. Instead of providing a factory for every shared multiplication, five in the case of Kit, we can instantiate the design with only one factory. This implies that the evaluation stage waits for the preprocessing stage. Essentially, this is a *bit-based paradigm*, where only one bit of the state is processed at a time. As a result, the preprocessing stage is five times smaller and the design is five times slower. Given that the preprocessing stage is not dominant in size for many instances of Kit, this design has not been implemented.

Summary of the designs. Table 1 summarizes the amount of S-boxes and factories of every implementation, and the number of cycles they take to execute.

3.2 Preprocessing stage

In the computing procedure for the Beaver multiplication [36], the auxiliary value $\langle \boldsymbol{c} \rangle$ is only needed *one cycle later* than its corresponding $\langle \boldsymbol{a} \rangle$ and $\langle \boldsymbol{b} \rangle$. However, a naive implementation of the preprocessing stage provides $\langle \boldsymbol{a} \rangle$, $\langle \boldsymbol{b} \rangle$ and $\langle \boldsymbol{c} \rangle$ at the same time. By tolerating a one-cycle lag of the auxiliary values $\langle \boldsymbol{c} \rangle$ with respect to $\langle \boldsymbol{a} \rangle$ and $\langle \boldsymbol{b} \rangle$ values, as depicted in Fig. 5, we save several storage elements at the boundary between evaluation and preprocessing stage. When the number of factories is large, this algorithm-independent optimization incurs a considerable resource saving.

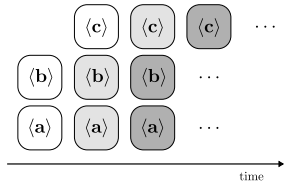


Fig. 5: The one-cycle lag of $\langle \boldsymbol{c} \rangle$ w.r.t $\langle \boldsymbol{a} \rangle$ and $\langle \boldsymbol{b} \rangle$

4 Implementation results and cost scalability

Our synthesis results are obtained with Synopsys Design Compiler N-2017.09 in conjunction with the freely available NANGATE 45nm Open Cell technology library [31]. The compilation is done with the `exact_map` option enabled and clock gating disabled to prevent cross-tile optimizations.

Table 2: Synthesis results for KECCAK-f[200]

KECCAK-f[200] in NANGATE 45nm ($\boldsymbol{m = 2}$)												
		AREA [kGE]								Rand.	f_{max}	FoM
		Evaluation				Preprocessing			Total			
Design	**Order**	χ	θ	State	Σ	Gen.	Ver.	Σ		[bpc]	[MHz]	[kbps/GE]
BLAZE	1	73.6	4.8	12.5	94.3	114.8	103.0	217.8	312.1	4400	806	25.8
	2	117.4	7.2	18.7	148.5	261.7	166.2	427.9	576.4	10800	806	14.0
	3	166.0	9.6	25.0	207.4	469.5	237.6	707.1	914.5	20000	751	8.2
FAST	1	36.8	4.8	11.1	56.1	57.4	51.5	108.9	165.0	2200	1333	28.9
	2	58.8	7.2	16.6	87.7	130.8	83.1	213.9	301.6	5400	1190	14.1
	3	83.1	9.6	22.2	121.6	234.7	118.8	353.5	475.2	10000	1190	8.9
FUR	1	9.2	4.8	13.2	27.9	14.3	12.9	27.2	55.1	550	1219	27.1
	2	14.7	7.2	19.9	42.7	32.7	20.8	53.5	96.2	1350	1098	14.0
	3	20.8	9.6	26.5	58.1	58.7	29.7	88.4	146.5	2500	1098	9.2
KIT	1	1.9	1.9	11.0	15.6	2.9	2.6	5.4	21.1	110	1315	12.1
	2	3.0	2.9	16.5	23.6	6.5	4.2	10.7	34.3	270	1162	6.6
	3	4.2	3.9	22.0	31.7	11.7	5.9	17.7	49.4	500	1176	4.6

Although the designs generally cover all seven KECCAK$-f[b]$ instances, for brevity, Table 2 reports the results for KECCAK-f[200], for variable side-channel protection orders. Recall that an implementation with d tiles achieves a side-channel protection order of $(d-1)$. To conduct a consistent comparison of the different designs, we fix $m = 2$ and warn the reader that this security parameter does not correspond to a satisfactory practical protection against fault attacks.

The relevant metrics are the area of the implementation, the randomness consumption, the maximum clock frequency f_{max}, the number of cycles n_c (see Table. 1) and a figure of merit (FoM) [25]. The area is reported in a hierarchical fashion so as to reveal how the total area is apportioned among the major parts of the design. The FoM jointly captures the performance in terms of speed and area and is given by $(b \cdot f_{max})/(A \cdot n_c)$, where A is the total area. As a measure of throughput per area (higher is better), the expression for the FoM directly follows from the reasoning that the throughput of KECCAK-f is proportional to the maximum clock frequency and the permutation width, but inversely proportional to the number of cycles.

Table 3: Comparison with side-channel only countermeasures. Note: the implementations of [21] are synthesized with UMC 90nm and clock gating

KECCAK-f[1600] in NANGATE 45nm ($\boldsymbol{m = 0}$)										
		AREA [kGE]						Rand.	f_{max}	Cycles
		Evaluation				Prep.	Total			
Order	**Design**	χ	θ	State	Σ			[bpc]	[MHz]	[/]
1	BLAZE	145.1	12.8	33.7	199.7	231.0	430.7	16000	892	25
	Parallel [21]	38.4	15.0	32.2	85.7	-	85.7	480	891	48
	Parallel-3sh [10]	40.6	19.2	56.8	116.6	-	116.6	4	592	25
	KIT	0.5	0.6	26.1	29.1	0.7	29.8	50	1538	10776
	Serial-Area [21]	0.4	0.4	14.5	15.7	-	15.7	-	850	3160
	Serial-3sh [10]	0.6	0.3	38.1	39.0	-	39.0	< 1	645	1625
2	BLAZE	235.2	19.2	50.5	317.1	449.3	766.4	28800	884	25
	Parallel [21]	114.0	22.5	51.1	188.1	-	188.1	4800	898	48
	KIT	0.7	1.0	39.1	43.7	1.4	45.1	90	1351	10776
	Serial-Area [21]	2.2	0.6	21.4	24.2	-	24.2	75	898	3160
KECCAK-f[200] in NANGATE 45nm ($\boldsymbol{m = 0}$)										
1	BLAZE	18.1	1.6	4.2	25.2	28.9	54.0	2000	892	19
	5-10-5 [1]	73.4	14.0	11.9	99.3	-	99.3	-	395.25	9
	6-6-6 [1]	44.6	11.3	14.2	70.1	-	70.1	-	436.7	9

4.1 Comparison with the literature

The literature already features protected implementations of KECCAK, although their protection scope is limited to side-channel analysis (SCA) in a weaker attacker model [1,10,21]. Table 3 compares these implementations with the ones introduced in this work, which are instantiated for $m = 0$, implying that the final MAC checking phase can be avoided and all implementations are one cycle faster. It should be stressed that this is not the intended setting for CAPA, but merely serves for comparison. When the fault protection capability is not used, one should indeed opt for another countermeasure. Although there are many trade-offs in the designs, the comparison features only the fastest and the smallest design of every source to cover the limits of high speed and low area. For this paper, the representative designs are BLAZE and KIT, which adopt the permutation width of the prior work they are compared to.

For the full permutation width $b = 1600$, BLAZE is nearly twice as fast as its competitors [10,21]. The price to pay is a larger area and a huge randomness consumption. For the the $b = 200$ instance, BLAZE has a competitive speed and significantly smaller area than the unrolled implementations of [1] at the expense of a substantially larger randomness cost.

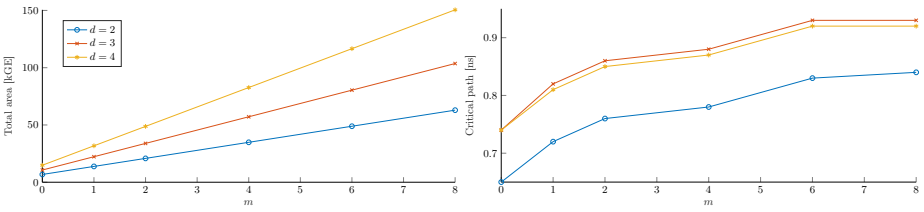
The first-order protected KIT design is clearly smaller than the Serial-3sh implementation [10] because the latter employs three shares. It would appear that the row-based efforts in the KIT design yield a (much) larger implementation than the Serial-Area implementation [21]. However, the difference can largely be

attributed to *(1) different technology libraries* and *(2)* that the synthesis results for Serial-Area are obtained with *clock gating enabled*. We verify this hypothesis as follows for the first-order secure design. Without counting multiplexers, accommodating the state using scan flip-flops (SFF) in NANGATE 45nm [31] already has an area cost of $8.0\frac{\text{GE}}{\text{SFF}} \cdot 1600 \cdot 2\text{SFF} = 25.6\text{kGE}$. This is already in the same order of magnitude as the first-order KIT design as reported in Table 3.

4.2 Cost scalability

Scaling with m. The implementation *area* is expected to scale linearly as a function of the fault security parameter m. Incrementally increasing m incurs an additional parallel computation in every arithmetic unit and an extra storage unit for every shared value $\langle \boldsymbol{x} \rangle$. The control overhead is not expected to scale with m. To corroborate this intuition, we gather experimental evidence for KIT, the design where the control logic has the largest relative size. Fig. 6a shows the results for KECCAK-f[200] with different values of d. Because of the near-perfect linear relationship, we take it as a given and fix m in the discussions that follow.

The *throughput* is affected, albeit slighty, by the value of m. The number of cycles is constant with respect to the fault security parameter m. Because the tags corresponding to different keys never interact, no significant increase in the critical path is to be expected either. However, Fig. 6b shows the contrary and reveals a monotonous increase of the critical path with respect to m. It can be attributed to the placement and routing of the cells.



(a) (b)

Fig. 6: (a) KECCAK-f[200] implementation size as function of m, for the KIT design. (b) The critical path increases monotonously with m

Scaling with d. The *area* of the evaluation stage scales linearly with the SCA protection parameter d. The preprocessing stage, on the other hand, scales asymptotically with d^2 due to the presence of the passively secure multipliers. Fig. 7 presents experimental evidence for these claims. The FUR and KIT designs are particularly interesting for large SCA protection orders as the offline part, i.e. the part of the implementation that scales with d^2, is relatively small. The impact on the *throughput* can be determined as follows. Similar to m, the

SCA security parameter d does not affect the number of cycles of the algorithm. To assess its influence on the critical path, consider Fig. 6b but now keeping m constant and varying d. There seems to be some dependency on d but there is no monotonous increase. The variations are likely due to placement and routing.

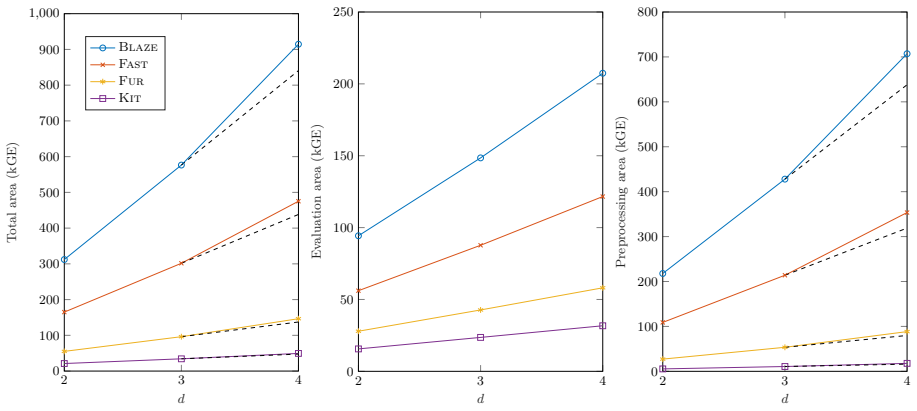
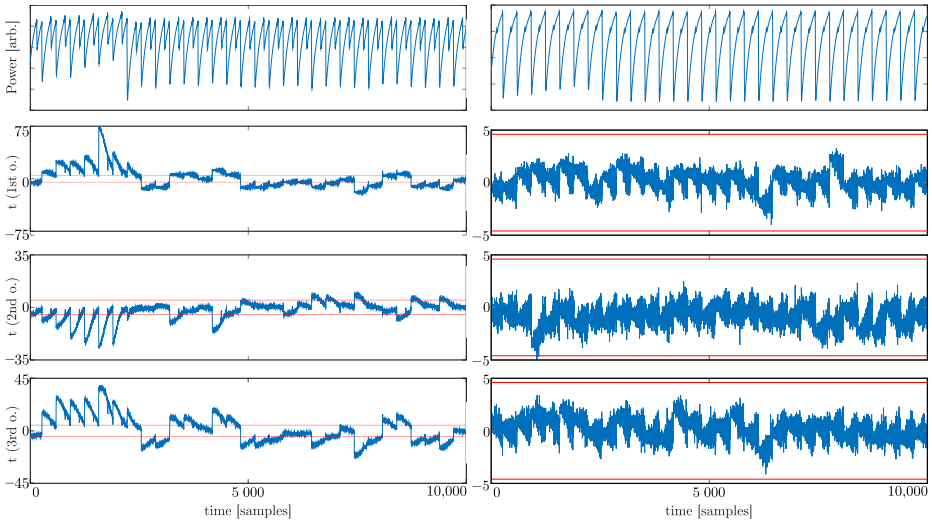


Fig. 7: Area of the designs (Keccak-$f[200]$) w.r.t the SCA parameter d, for $m = 2$. From left to right: total area, evaluation stage, preprocessing stage

Scaling with b. As the Keccak-f permutation width increases, a larger state has to be processed every round. As a result, the designs either become substantially larger (Blaze, Fast) or suffer a considerable throughput penalty (Fur, Kit). Table 4 allows to interpret the influence of the permutation width on the *area* of the implementation. The last column features the interpolated numbers. In particular, the linear coefficient in b is of importance for this experiment. As expected, the slice- and row-based designs scale much less dramatically than the designs that consider the entire state or half of the state every round. The critical path is unaffected by b. This makes sense as the fundamental operations occur at the bit-level and remain unchanged. The *throughput* scaling can then simply be deduced from the number of cycles in Table 1.

Table 4: Area scaling of the designs for Keccak-$f[b]$ w.r.t. b

Area [kGE] of Keccak-$f[b]$ **for $d = 2, m = 2$**					
Design	$b = 200$	$b = 400$	$b = 800$	$b = 1600$	Interpolation
Blaze	312.1	623.9	1247.5	2494.6	$312.1 + \mathbf{1.56}(b - 200)$
Fast	165.0	329.7	659.1	1317.8	$165.0 + \mathbf{0.82}(b - 200)$
Fur	55.1	72.6	107.5	177.2	$55.1 + \mathbf{0.087}(b - 200)$
Kit	21.1	30.9	50.5	89.8	$21.1 + \mathbf{0.049}(b - 200)$



(a) Masks off (b) Masks on

Fig. 8: Results of the non-specific leakage detection (t-test) for the KIT design of KECCAK-f[200], where $d = 3$ and $m = 2$. Top row: average power trace (arbitrary units). Second until last row: resp. 1^{st}, 2^{nd} and 3^{rd} order t-test

5 Security evaluation

5.1 Side-channel resistance

To evaluate the side-channel resistance of the presented designs, we employ the *Test Vector Leakage Assessment* (TVLA) method [4]. Specifically, we consider a *non-specific* leakage detection test and adopt a t-test threshold of $|t| = 4.5$. The evaluation platform is a Sakura-G board, equipped with two 45nm Xilinx Spartan-6 FPGAs that respectively host the cryptographic implementation and the control unit. By design, the board isolates the power supplies of the FPGAs, mitigating the noisy influence of the control unit on the power consumption measurements. The programming files for the cryptographic unit are obtained with the `keep_hierarchy` constraint to avoid that sources of out-of-model leakage are added by the synthesis of the design. The implementation is clocked at 3 MHz and the masks are produced by a KECCAK-f[1600]-based PRNG.

The evaluation is performed on the KECCAK-f[200] instance of the KIT design. This design should exhibit the fastest evidence of leakage as it only features a relatively small preprocessing phase, which contributes as a noise source in the context of this experiment. Because the implementations are general in b, the security claims carry over to the other KECCAK-$f[b]$ instances. The implementation is parametrized with $d = 3$ tiles (offering a second-order secure design) and a fault security parameter $m = 2$. The specific value of m is unimportant, but is

non-zero to ensure that the synthesis and implementation tools do not trim the preprocessing stage and MAC checks from the implementation.

To construct the power traces, an oscilloscope captures the voltage drop over a 1Ω shunt resistor at $1GS/s$ for 10000 samples. The traces correspond to the latter half of the ρ mapping and the first two slices of χ, obtaining a representative part of the round function, with linear and nonlinear operations. To ensure that the test setup is able to detect leakage, all masks are turned off in the first iteration of the experiment. In the subsequent iteration, the masks are turned on. The leakage reduction from the first experiment to the second on can then be fully attributed to the Capa countermeasure.

The experiment where the masks are disabled, shown in Fig. 8a, shows serious leakage. Already after the first batch of 15000 traces, $|t|$ amply exceeds the threshold of $|t| = 4.5$. This observation validates that the measurement setup is reliable. When activating the masks, the t-test reveals no first-, second- or third-order leakage when presented 80 million traces (Fig. 8b). Although we do not claim that the implementation with three shares is secure against third-order attacks, no leakage is apparent from the statistical evidence contained within the supplied traces. This can be attributed to the measurement noise.

In Fig. 8b, the second-order t-test can be seen to approach the threshold value of $t = \pm 4.5$. To provide reassurance that this artefact stems from statistical variations as opposed to genuine leakage, Fig. 9 shows the maximum $|t|$-values over time. It can be seen to fluctuate around the threshold but no steady increase in its value is recognizable. In conclusion, the experiment does not provide evidence of second-order leakage.

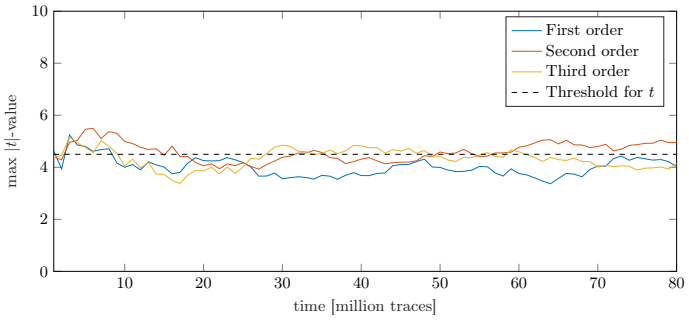


Fig. 9: Maximum t-test value over time

5.2 Fault resistance

Resistance against fault attacks is difficult to evaluate as currently no established formal verification procedures exist. For a theoretical discussion on the fault resistance of Capa, we refer to [36]. While not conclusive, we gain confidence in the implementation by experimentally verifying the fault detection probability.

Table 5: Experimental fault resistance results

	$\boldsymbol{m} = 2$	$\boldsymbol{m} = 4$	$\boldsymbol{m} = 6$	$\boldsymbol{m} = 8$
# valid $\langle \boldsymbol{f} \rangle$	32	512	8192	131072
# detected $\langle \boldsymbol{f} \rangle$	24	480	8064	130560

Modelling the attacker. Recall that CAPA considers the *tile-probe-and-fault* model, allowing the adversary to fault arbitrary bits in the implementation, given that at least one tile is not faulted nor probed. The adversarial goal is to introduce a fault f in the computations. To this end, she guesses a valid shared representation $\langle \boldsymbol{f} \rangle$, denoted a *fault vector*, consisting of $d(1 + m)$ bits. This results in $2^{d(1+m)}$ possible fault vectors. In $GF(2)$, half of these correspond to $f = 0$ and are hence excluded. The attacker can simultaneously fault at several interesting locations in the implementation (preprocessing, the linear θ mapping, the MAC check and the S-box) but will stick to one guess of the MAC key; otherwise she will be detected with probability one. To cover the DFA attacks on KECCAK [2], we inject the faults in the penultimate round. We modify the HDL implementation in the targeted modules to introduce the fault vectors as additive differences as done in [15]. We simulate the design with GHDL 0.36-dev.

Experimental results. We experimentally validate the claimed fault detection probability for the first-order ($d = 2$) KECCAK-f[200] KIT implementation with $m \in \{2, 4, 6, 8\}$. The results can then be extrapolated to larger m as both CAPA and the KIT implementation itself are generically scalable in m. For these relatively small values of m, we need not follow a probabilistic approach. The fault coverage can be trivially parallelized and we can *exhaustively* cover all valid fault vectors in a few hours. This deterministic experiment is successful if the fraction of detected faults is *exactly* equal to $1 - 2^{-m}$. Table 5 covers the experimental results and demonstrates that the implementation is sound.

6 Conclusion

Following the CAPA countermeasure methodology, this paper reports the first KECCAK implementations with resistance against combined side-channel and fault attacks. The fastest design competes with or even outperforms the state of the art in side-channel protected designs. As a drawback, the area and randomness requirements are prohibitively large. The smaller designs of this work have more attainable requirements, but incur a considerable throughput penalty. All four approaches are general in the KECCAK permutation width b, and scalable in the number of tiles d and the fault security parameter m. We have presented and validated the scaling laws as a function of these parameters. An advanced leakage detection test on the most intricate implementation of this paper has validated our confidence in its SCA resistance. The soundness of the implementation with respect to fault attacks has been supported by simulation.

Acknowledgements. The authors would like to thank the COSADE reviewers for their helpful comments. This work was supported in part by the Research Council KU Leuven: C16/15/058 and by the NIST Research Grant 60NANB15D346. Lauren De Meyer is funded by a PhD fellowship of the Fund for Scientific Research - Flanders (FWO). Antoon Purnal would like to thank Vincent Rijmen and Ingrid Verbauwhede for supervising the master's thesis that led to this paper.

References

1. Arribas, V., Bilgin, B., Petrides, G., Nikova, S., Rijmen, V.: Rhythmic Keccak: SCA Security and Low Latency in HW, journal = IACR Trans. Cryptogr. Hardw. Embed. Syst., volume = 2018, number = 1, pages = 269–290, year = 2018,
2. Bagheri, N., Ghaedi, N., Sanadhya, S.K.: Differential fault analysis of SHA-3. In: Biryukov, A., Goyal, V. (eds.) Progress in Cryptology – INDOCRYPT 2015. pp. 253–269. Springer International Publishing, Cham (2015)
3. 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)
4. Becker, G., Cooper, J., De Mulder, E., Goodwill, G., Jaffe, J., Kenworthy, G., Kouzminov, T., Leiserson, A., Marson, M., Rohatgi, P., et al.: Test vector leakage assessment (TVLA) methodology in practice. In: International Cryptographic Module Conference. vol. 1001, p. 13 (2013)
5. Bertoni, G., Breveglieri, L., Koren, I., Maistri, P., Piuri, V.: Error analysis and detection procedures for a hardware implementation of the advanced encryption standard. IEEE Trans. Computers **52**(4), 492–505 (2003)
6. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Keccak sponge function family main document. Submission to NIST (Round 2) **3**(30) (2009)
7. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Building power analysis resistant implementations of Keccak. In: Second SHA-3 candidate conference. vol. 3, p. 2. Citeseer (2010)
8. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G., Van Keer, R.: CAESAR submission: Ketje v2 (2015)
9. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G., Van Keer, R.: CAESAR submission: Keyak v2 (2015)
10. Bilgin, B., Daemen, J., Nikov, V., Nikova, S., Rijmen, V., Assche, G.V.: Efficient and first-order DPA resistant implementations of Keccak. In: Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers. pp. 187–199 (2013)
11. Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., Rijmen, V.: Higher-order threshold implementations. In: Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II. pp. 326–343 (2014)
12. Boneh, D., DeMillo, R.A., Lipton, R.J.: On the importance of checking cryptographic protocols for faults (extended abstract). In: Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding. pp. 37–51 (1997)

13. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards sound approaches to counteract power-analysis attacks. In: Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings. pp. 398–412 (1999)
14. De Cnudde, T., Nikova, S.: More efficient private circuits II through threshold implementations. In: Fault Diagnosis and Tolerance in Cryptography (FDTC), 2016 Workshop on. pp. 114–124. IEEE (2016)
15. De Meyer, L., Arribas, V., Nikova, S., Nikov, V., Rijmen, V.: M&M: Masks and Macs against physical attacks. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2019**(1), 25–50 (2019)
16. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic analysis: Concrete results. In: Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings. pp. 251–261. No. Generators (2001)
17. Gierlichs, B., Schmidt, J., Tunstall, M.: Infective computation and dummy rounds: Fault protection for block ciphers without check-before-output. In: Progress in Cryptology - LATINCRYPT 2012 - 2nd International Conference on Cryptology and Information Security in Latin America, Santiago, Chile, October 7-10, 2012. Proceedings. pp. 305–321 (2012)
18. Goubin, L., Patarin, J.: DES and differential power analysis (the "duplication" method). In: Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99, Worcester, MA, USA, August 12-13, 1999, Proceedings. pp. 158–172 (1999)
19. 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 (2016)
20. Groß, H., Mangard, S., Korak, T.: 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. pp. 95–112 (2017)
21. Groß, H., Schaffenrath, D., Mangard, S.: Higher-order side-channel protected implementations of Keccak. In: Euromicro Conference on Digital System Design, DSD 2017, Vienna, Austria, August 30 - Sept. 1, 2017. pp. 205–212 (2017)
22. Guo, X., Karri, R.: Recomputing with permuted operands: A concurrent error detection approach. IEEE Trans. on CAD of Integrated Circuits and Systems **32**(10), 1595–1608 (2013)
23. Ishai, Y., Prabhakaran, M., Sahai, A., Wagner, D.A.: Private circuits II: keeping secrets in tamperable circuits. In: Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings. pp. 308–327 (2006)
24. 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. pp. 463–481 (2003)
25. Jungk, B., Apfelbeck, J.: Area-efficient FPGA implementations of the SHA-3 finalists. In: 2011 International Conference on Reconfigurable Computing and FPGAs, ReConFig 2011, Cancun, Mexico, November 30 - December 2, 2011. pp. 235–241 (2011)

26. Karpovsky, M.G., Kulikowski, K.J., Taubin, A.: Robust protection against fault-injection attacks on smart cards implementing the advanced encryption standard. In: 2004 International Conference on Dependable Systems and Networks (DSN 2004), 28 June - 1 July 2004, Florence, Italy, Proceedings. pp. 93–101 (2004)
27. Kim, C.H., Quisquater, J.: Fault attacks for CRT based RSA: new attacks, new results, and new countermeasures. In: Information Security Theory and Practices. Smart Cards, Mobile and Ubiquitous Computing Systems, First IFIP TC6 / WG 8.8 / WG 11.2 International Workshop, WISTP 2007, Heraklion, Crete, Greece, May 9-11, 2007, Proceedings. pp. 215–228 (2007)
28. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings. pp. 104–113 (1996)
29. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings. pp. 388–397 (1999)
30. Kulikowski, K.J., Karpovsky, M.G., Taubin, A.: Robust codes and robust, fault-tolerant architectures of the advanced encryption standard. Journal of Systems Architecture **53**(2-3), 139–149 (2007)
31. NANGATE: California. 45nm open cell library. Available at http://www.nangate.com (2008)
32. Nikova, S., Rechberger, C., Rijmen, V.: Threshold Implementations Against Side-Channel Attacks and Glitches, pp. 529–545 (2006)
33. Nikova, S., Rijmen, V., Schläffer, M.: Secure hardware implementation of non-linear functions in the presence of glitches. In: Information Security and Cryptology - ICISC 2008, 11th International Conference, Seoul, Korea, December 3-5, 2008, Revised Selected Papers. pp. 218–234 (2008)
34. Patel, J.H., Fung, L.Y.: Concurrent error detection in ALU's by recomputing with shifted operands. IEEE Trans. Computers **31**(7), 589–595 (1982)
35. 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. pp. 764–783 (2015)
36. Reparaz, O., De Meyer, L., Bilgin, B., Arribas, V., Nikova, S., Nikov, V., Smart, N.P.: CAPA: the spirit of Beaver against physical attacks. In: Advances in Cryptology - CRYPTO '18, 38th Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2018 (2018)
37. 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. pp. 413–427 (2010)
38. Schneider, T., Moradi, A., Güneysu, T.: ParTI - towards combined hardware countermeasures against side-channel and fault-injection attacks. In: Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II. pp. 302–332 (2016)
39. van Woudenberg, J.G.J., Witteman, M.F., Menarini, F.: Practical optical fault injection on secure microcontrollers. In: 2011 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2011, Tokyo, Japan, September 29, 2011. pp. 91–99 (2011)
40. Yen, S., Joye, M.: Checking before output may not be enough against fault-based cryptanalysis. IEEE Trans. Computers **49**(9), 967–970 (2000)