Towards Evaluating DPA Countermeasures for KECCAK on a Real ASIC

Michael Muehlberghuber[1], Thomas Korak[2], Philipp Dunst[2], and Michael Hutter[3,⋆]

[1] Integrated Systems Laboratory (IIS), ETH Zurich, Gloriastrasse 35, 8092 Zurich, Switzerland
mbgh@iis.ee.ethz.ch

[2] Institute for Applied Information Processing and Communications (IAIK), Graz University of Technology, Inffeldgasse 16a, 8010 Graz, Austria
thomas.korak@iaik.tugraz.at, p.dunst@gmx.net

[3] Cryptography Research, 425 Market Street, San Francisco, CA 94105, USA
michael.hutter@cryptography.com

Abstract. We present ZORRO, a taped-out ASIC hosting three distinct authenticated encryption architectures based on the SPONGEWRAP construction. All designs target resource-constrained environments such as smart cards or embedded devices and therefore, have been protected against DPA attacks while keeping low-area as the most important design goal in mind. Each of the three architectures contains masking and hiding countermeasures. They solely differ with regard to the implemented secret-sharing scheme. While the first design is based on a 3-share threshold implementation (TI), which does not fulfill the uniformity property, the other two make use of the 3-share approach with re-masking and the 4-share approach as proposed by Bilgin et al. Our smallest, provable first-order DPA secure KECCAK implementation requires only 14.5 kGE (which is less than half of the size of related work) and contains both front-end and back-end design overheads. Moreover, we present first DPA results of the ZORRO ASIC by comparing hiding and masking countermeasures. We were able to recover the cipherkey from a masking-secured TI implementation based on three shares with about 70 000 power traces.

Keywords: Duplex construction, SPONGEWRAP, threshold implementation, side-channel attacks, DPA, low-area hardware, ASIC.

1 Introduction

Confidentiality and authenticity of data are among the most important cryptographic services required to transfer data securely over public communication channels. The former is commonly achieved by symmetric encryption algorithms while the latter is often obtained by message authentication codes (MACs). These cryptographic primitives have been treated independently in the past,

⋆ This work was done while the author was with Graz University of Technology.

which led to inefficient solutions and severe security problems [9,11]. For this reason, researchers have started to develop new hybrid algorithms that offer the desired service of authenticated encryption (AE), for instance, as part of the on-going CAESAR competition [1].

The SpongeWrap construction [3] uses the underlying permutation of Keccak—the winner of the NIST SHA-3 competition [6] in 2012—in order to realize an AE system. Implementations using Keccak-f in a keyed mode[4] such as SpongeWrap, necessarily require protection against implementation attacks such as Differential Power Analysis (DPA) [15]. Since especially smart cards and embedded systems are usually accessible by a broader mass of people, countermeasures like *hiding* or *masking* techniques are mandatory for such devices nowadays. The authors of Keccak proposed to implement a secret sharing technique to protect keyed Keccak instances [2,7]. This technique is based on the idea to divide key-dependent intermediate values into unique parts (so-called shares) and to re-combine them after the processing. In order to achieve first-order DPA resistance, this sharing needs to fulfill three properties: correctness, non-completeness, and uniformity [17]. Interestingly, Bilgin et al. [8] reported that the implementation in [2,7] does not fulfill the uniformity property and is therefore not provable secure against first-order DPA attacks. As a countermeasure, they proposed to inject fresh random bits in a 3-share implementation or to add an additional share (4-share version) that avoids the need of fresh randomness.

This work presents first results of an actually "taped-out" application-specific integrated circuit (ASIC), called Zorro (our chip is not to confuse with the block-cipher of Gérard et al. [12] proposed at CHES 2013). Zorro hosts three distinct hardware architectures for SpongeWrap-based authenticated encryption, secured against DPA. The chip is intended to be used as a fully flexible evaluation platform for determining the effectiveness of hiding and masking countermeasures in a real ASIC. Therefore, the three architectures solely differ with regard to the realized masking technique. While the first design makes use of the 3-share approach proposed by Bertoni et al. [2,7], the latter two utilize the 3-share implementation with re-masking and the 4-share approach presented by Bilgin et al. [8], which both fulfill the uniformity property. Moreover, each of the three architectures contains hiding countermeasures, which can be switched on and off at will. The main fields of application for Zorro are resource-limited environments such as smart cards, embedded systems, or RFID-based devices, which is why low-area was our most important design goal. Zorro was fabricated in a 180 nm CMOS process technology by UMC and the smallest of the three architectures requires only 14.5 kGE. This represents the smallest reported masked Keccak ASIC implementation to date. Beside the un-keyed Keccak implementations available in literature [4,14,20], the smallest reported masking-secured designs so far require more than 30 kGE [2,7,8].

Moreover, we are the first to present DPA results targeting Keccak implementations on a fabricated ASIC chip. We provide first DPA results of the

[4] This mode involves a secret key that needs to be protected against implementation attacks. It is used in, e.g., stream encryption or authenticated encryption modes.

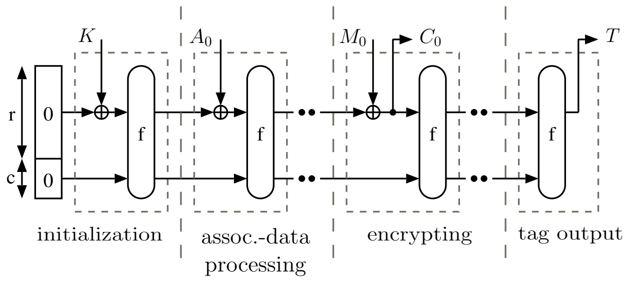


Fig. 1. The SpongeWrap construction.

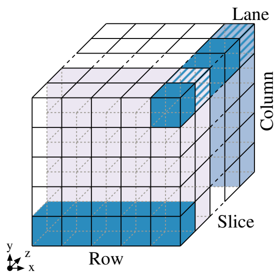


Fig. 2. Keccak state.

unprotected, the hiding-secured, and the three share threshold implementation (TI). A higher-order DPA attack against the masked implementation succeeds with about 70 000 measurements. In order to reach a comparable security level with the hiding countermeasure, an impractical number of 240 dummy rounds needs to be inserted, what equals ten times the number of rounds (24) for one Keccak-f permutation. Future work will consist in a detailed comparison of the three threshold implementations on the ASIC, requiring a huge amount of measurements which are not available yet.

The remainder of this paper is structured as follows. In Section 2, we give a brief introduction to the authenticated encryption mode SpongeWrap. Section 3 presents the hardware architecture of Zorro. Implementation and power-analysis results are given in Section 4 and finally a discussion about the results and future work is provided in Section 5.

2 The SpongeWrap Construction

The core element of SpongeWrap is the Keccak-f permutation [6]. The most prominent application of Keccak-f is its use in a sponge construction [5] to build the hash algorithm Keccak, which has recently been presented as a new draft for the upcoming SHA-3 standard by NIST [19]. However, Keccak-f can also be used to form several other cryptographic primitives [3], including the AE mode SpongeWrap. In the following, a brief introduction to the SpongeWrap construction and the Keccak-f permutation is given. For an in-depth discussion about the two primitives, we refer the reader to [3] and [6].

The SpongeWrap construction. Fig. 1 illustrates the SpongeWrap mode, which uses a duplex construction [3] to create an AE scheme. It can be subdivided into four phases: an *initialization* phase, an *associated-data processing* phase, an *encryption* phase, and a *tag-generation* phase. During the *initialization*, the state is cleared and loaded with the cipherkey K by a call to the permutation f. After that, the SpongeWrap object is able to receive data for *wrapping* associated data blocks A_i (authenticated only) and plaintext blocks M_j (authenticated and encrypted) to retrieve the ciphertext blocks C_j and the corresponding authentication tag T. The respective decryption process is known as *unwrapping* and basically swaps plaintext and ciphertext blocks and compares the received authentication tag with the recomputed one. If the two tags do not match, an error will be dumped, but no plaintext will be provided.

The KECCAK-*f* permutation. KECCAK-*f* operates on a state with a fixed size of b bits. This state consists of two parts: r (bitrate) and c (capacity), where r specifies the number of input bits, which are processed in one iteration and therefore relates to the speed of the computation. The last c bits of the state determine the attainable security level of the construction, i.e., $c = b - r$. The authors of KECCAK defined KECCAK-*f* for the following seven state sizes: $b = 25 \times w$, where $w = 2^{\ell}$ and ℓ ranges from 0 to 6. The state is organized as a $5 \times 5 \times w$ matrix with three dimension coordinates (x,y,z). We call a set of w bits with given (x,y) coordinates a *lane*, a set of 5 bits with given (y,z) coordinates a *row*, 5 bits with given (x,z) coordinates a *column*, and the 5×5 matrix for a given (z) coordinate a *slice* (see Fig. 2). The KECCAK-*f* function further consists of $12 + 2\ell$ rounds that are made up of five steps:

θ **:** Used to integrate diffusion by a linear mixing layer (the parity of two nearby columns is added to each column).
ρ **:** Inter-slice dispersion (all lanes are rotated by a defined offset).
π **:** Breaking horizontal/vertical alignment (the 25 lanes are transposed in a fixed pattern).
χ **:** The non-linearity part of KECCAK-*f* (the 5 bits of each row are combined using AND gates and inverters and the shifted result is added to the row).
ι **:** A w-bit round constant is added (XORed) to a single lane.

3 Hardware Architecture of ZORRO

We intend to use ZORRO as an evaluation platform for investigating the quality of DPA countermeasures for an AE system based on the KECCAK-*f*[1600] permutation. Our main goal was to build an ASIC, providing different types of *masking* and *hiding* techniques. As pointed out by Bilgin et al. [8], a first-order DPA-secure KECCAK design, which is based on a three-share threshold implementation (without re-masking), does not fulfill the *uniformity property* [17,18] and thus, is not provable secure against first-order DPA attacks. Hence, we decided to place three distinct architectures on ZORRO, which only differ with regard to the implemented masking scheme. The first design is based on a three-share approach as proposed by Bertoni et al. [7]. The second and third architectures make use of the threshold implementation improvements presented by Bilgin et al. [8], namely a three-share design using re-masking and a four-share architecture. Fig. 3 shows a block diagram of the top-level design entity, including the three distinct architectures named *3-Share*, *3-Share**, and *4-Share*.

In order to assure that meaningful power measurements can be taken from each distinct architecture separately, the *Clock Enable* entity contains clock gating cells, which enable the clock only for the actually selected entity. Moreover,

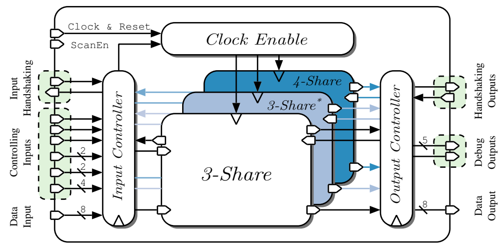


Fig. 3. Top-level architecture of Zorro.

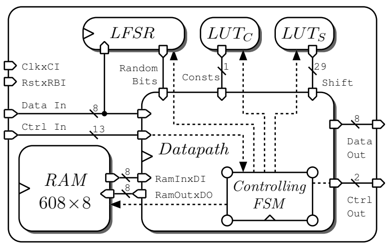


Fig. 4. Design of the *3-Share* entity.

the *Input Controller* forwards the input signals solely to the currently activated entity, thereby avoiding any logical changes in combinational paths within the deactivated architectures. With this setup, we are able to obtain meaningful power measurements of each design without significant noise from the deactivated units with regard to their dynamic power consumption. The *Output Controller* is responsible for forwarding the output signals of the respective unit once an input data block has been processed. Using a couple of debug outputs, Zorro provides additional information about currently ongoing internal processes. Data to and from the chip can be transmitted via an eight bit data bus, controlled by a four way handshaking protocol. Each of the three architectures by itself can either operate in encryption or decryption and offers four different modes of operation:

Normal Mode: The normal mode represents the default mode in which no DPA countermeasures are activated. Hence, only one third (for the three-share based architectures) respectively one fourth (for the four-share based design) of the state-storing RAMs is actually used. Measurements based on this mode serve as a reference for the protected alternatives.

Hiding Mode: Running in this mode, Zorro uses two hiding countermeasures in order to circumvent DPA attacks. First, the user can choose how many *dummy operations* should be executed during processing a single input block. Using a control signal, up to 15 dummy operations can be initiated, each of them representing a full round of the Keccak-f[1600] permutation. Second, all three architectures can shuffle their computations by varying between eight different read/write addresses when accessing the RAM.

Masked Mode: When operating Zorro in this mode, en-/decryption is performed using masking countermeasures in order to prevent DPA attacks.

Secure Masked Mode: In this mode, both hiding and masking countermeasures are activated and hence, this represents the most secure way on how to operate Zorro with regard to its DPA security.

3.1 *3-Share*, *3-Share**, and *4-Share* Architectures

Since the *3-Share*, *3-Share**, and *4-Share* hardware architectures differ only very slightly, we will further on solely discuss the *3-Share* version and point out the differences to the other two architectures if necessary.

We aimed to design a low-area, DPA-secure authenticated encryption system based on Keccak. Because of these goals, the area density of the memory required to store the Keccak state is of utmost importance. Moreover, the implemented secret sharing countermeasure works on the algorithmic level, and thus the required memory for the state increases with each share. Therefore, we favored a random-access memory (RAM) macro cell over their standard cell counterparts to store the state, which offers a better bit-per-area density. We store both the round constants of the ι function and the shift offsets of the ρ function in look-up tables (LUTs). Fig. 4 illustrates the uppermost hierarchy level of the *3-Share* entity, including the state RAM, the LUTs, and the datapath entity, which gets controlled by a finite-state machine (FSM). Moreover, Fig. 4 shows the linear-feedback shift register (LFSR), which is constructed by the primitive polynomial $x^{32} + x^7 + x^3 + x^2 + 1$. The initialization of the LFSR is done with an external seed. Its output is used on the one hand for determining whether to perform a dummy operation or not, and on the other hand for generating the random bits required for the re-masking in the *3-Share** architecture. Overall, 42 random bits are required per input block (39 for the dummy operation conditions and three for the shuffling of the RAM addresses).

The *3-Share* architecture contains a 608×8 RAM (cf. Fig. 4) for storing the state and the shares. Basically, a secret sharing scheme for Keccak based on three shares would require only 4 800 bits (three times the state size). We use the additional eight bytes of the RAM as inputs for the dummy operations during the hiding mode and therefore, keep these memory locations uninitialized. Thereby, none of the dummy operations computes on the actual payload of the chip and hence, no correlated power figures should be observed.

For the initial masking of the *3-Share* (*4-Share*) entity, the chip receives 3 200 (4 800) random bits to initialize two (three) shares followed by the plaintext. The last share equals the XOR-sum of the already initialized shares with the plaintext. The implementation of the Keccak-f[1600] permutation is based on a combined lane and slice processing, similar to that proposed by Pessl and Hutter [20]. Fig. 5 shows the architecture of the *Datapath* unit of the *3-Share* entity. We use the *SubState* register to buffer lanes and slices currently being processed.

RAM allocation. As proposed by Bertoni et al. [4], storing the bits of lanes and slices in an interleaved form allows efficient processing of the data when choosing a small datapath width, meanwhile keeping the size of the required buffer register at a minimum. We also make use of this technique and store four bits of two slices in each RAM word (i.e., two bits of four lanes). Since we need four lanes at a time, this results in a buffer register of 256 bits. Unfortunately, the state consists of 25 lanes and thus, not all lanes can be stored in this interleaved form. We decided to store the first lane in a linear way, as this lane is not influenced by the ρ operation and hence, can be skipped for this function. Although, based on this memory allocation we waste a negligible amount of clock cycles when loading data of the first lane, we can keep the size of the *SubState* register comparatively small. In order to avoid switching back and forth between slice-based and lane-

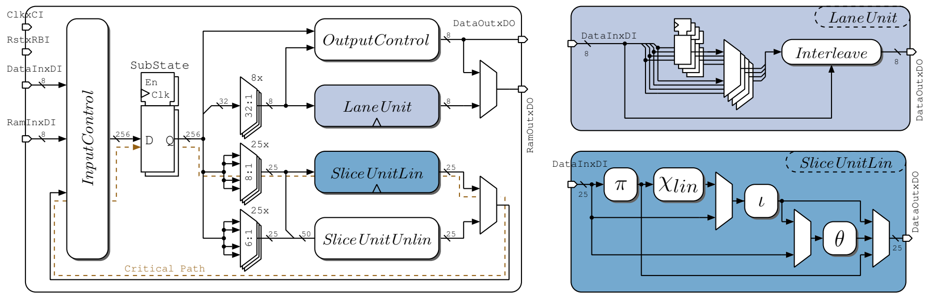


Fig. 5. Datapath of the *3-Share* entity (controlling signals omitted).

based operations as much as possible, we make use of the same rescheduling approach proposed in [20], where they distinguish between the following three different types of "rounds":

$$R_1 = \theta \times \rho \qquad R_{2\ldots24} = \pi \times \chi \times \iota \times \theta \times \rho \qquad R_{25} = \pi \times \chi \times \iota \tag{1}$$

Round operations. When ZORRO operates in normal mode, the four slice-based round functions of KECCAK-f (θ, π, χ, and ι) are exclusively calculated in the *SliceUnitLin* within a single clock cycle for a whole slice. The applied round schedule requires to calculate the result of θ, $\pi \times \chi \times \iota \times \theta$, and $\pi \times \chi \times \iota$. As illustrated on the bottom-right of Fig. 5, all three operations can be accomplished within the *SliceUnitLin* with the use of bypass multiplexers. Calculations of the linear round functions of the KECCAK-f permutation are equal for both the normal mode and the masking-secured modes. Here, each share can be computed in sequential order (e.g., in R_1 the theta step is performed three (four) times sequentially in order to process the three (four) shares). Due to the fact that the non-linear χ function requires inputs from more than one share, the processing of this function slightly differs. For the hardware implementation of the *3-Share* architecture, we follow the approach presented by Bertoni et al. [7] and compute the result for two input slices in a single cycle within the *SliceUnitUnlin* entity. For the lane-based operation (ρ), we aimed at calculating its output byte-by-byte. This allows us to combine it with the RAM write operation. Thanks to the chosen RAM allocation, multiples of two-bit-wide shift operations of lanes can easily be accomplished with the addressing of the memory. The special storage structure provides information about four lanes per RAM word (byte) and the *SubState* register can hold up to four lanes simultaneously. Unfortunately, each lane has a different shift offset. Hence, different bit couples of the buffered lanes must be taken to compensate the differences between the offsets. The different compensation offsets can be precalculated and are stored in the LUT_S entity (see Fig. 4) for each lane quadruple. With these precalculated values, multiples of two-bit-wide shift operations, and the offset between the different lanes can be compensated. The leftover is a possible shift by one bit. Therefore, 4 one-bit-registers with surrounding multiplexers are used. If a lane is shifted by one bit,

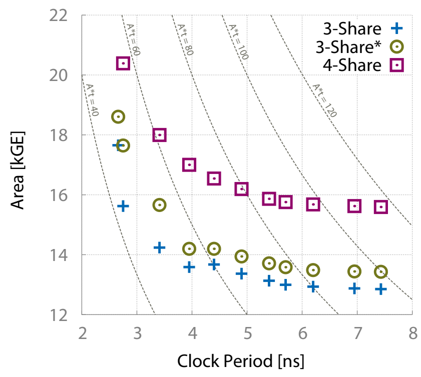


Fig. 6. AT plot of ZORRO's three different architectures obtained after synthesis.

Table 1. Area breakdown of the ZORRO ASIC (synthesis results at 5 ns).

Component	**Area [GE]**	**Area [%]**
3-Share	13'370	30.5
Datapath & FSM	7'300	16.7
RAM	4'680	10.7
LFSR	300	0.7
SliceUnitLin	480	1.1
Others	610	1.3
3-Share*	13'940	31.8
4-Share	16'190	37.0
I/O Interface	320	0.7
ZORRO Total	43'820	100.0

the high bit of the chosen bit couple is stored in the one bit register. The low bit is shifted one bit to the left and the old content of the one bit register is used as the new low bit. This is done for each bit couple of the buffered lanes. The result is stored back to the RAM in interleaved form. The responsible unit for the lane-based operation is called *LaneUnit* (cf. Fig. 5).

4 Results

The results of our work are twofold. First, we present our implementation results of ZORRO and provide actual ASIC performance numbers of the *3-Share*, *3-Share**, and *4-Share* design. Second, we present first practical results of DPA investigations on our AE system using power traces obtained from the real chip.

4.1 Hardware Figures of ZORRO

We used VHDL in order to code the RTL model of ZORRO and Mentor Graphics' ModelSim version 10.2a to verify its functional correctness. Synthesis results were obtained from Synopsys' Design Compiler version 2012.06 for a mature 180 nm CMOS technology by UMC. The designs were synthesized using a standard cell library by Faraday Technologies under typical case conditions and backend design steps were accomplished using SoC Encounter from Cadence. Area results will be given in terms of gate equivalents (GEs), for which one GE equals the size of a two-input NAND gate of the utilized standard cell library (= 9.3744 µm^2).

In order to provide a fair comparison between the results of ZORRO and related work as well as meaningful numbers for an actual chip to be taped out, we present two different area numbers. On the one hand, we provide synthesis results without considering any Design for Testability (DFT) techniques.[5] On

[5] Note that such numbers can vary significantly compared to the actual area figures of a finalized chip ready for tapeout, depending on the implemented design.

Table 2. Comparison of ZORRO with related ASIC designs (synthesis results).

Source	**Techn.** [nm]	**Area** [GE]	f_{max} [MHz]	**Perf.**[†] [Cycles]
Designs w/o DPA Countermeasures				
Pessl and Hutter [20][‡]	130	5'522	61	22'570
Bilgin et al. [8][§]	180	10'800	555	1'600
ZORRO in Normal Mode[‡]	180	13'370	200	21'888
3-Share-Secured Designs w/o Re-Masking				
Bertoni et al. [7][§]	130	95'000	200	72
ZORRO *3-Share* Architecture[‡]	180	13'370	200	113'184
3-Share-Secured Designs w/ Re-Masking				
Bilgin et al. [8][§]	180	33'100	553	1'625
ZORRO *3-Share** Architecture[‡]	180	13'940	200	113'184
4-Share-Secured Designs				
Bilgin et al. [8][§]	180	43'100	572	1'600
ZORRO *4-Share* Architecture [‡]	180	16'190	200	149'640

[†] KECCAK-f permutation [‡] Block size of 1088 bits [§] Block size of 1024 bits

the other hand, we include the area numbers after all backend design steps have been successfully accomplished and therefore the designs include DFT circuitries for RAM tests as well as scan flip-flops to enable automated test pattern generation (ATPG). Fig. 6 provides an area/time (AT) plot of the synthesis results of the three different architectures. Based on the isolines, indicating a constant AT product, it can be observed that for a clock period below 4 ns, the resulting area of each architecture increases significantly. Moreover, we decided to spend some room for the upcoming backend run and therefore, chose a maximum frequency of 200 MHz for ZORRO. The critical path of the design runs through the *SliceUnitLin* entity, highlighted using a dashed line in Fig. 5. From Fig. 6 it can be seen that the area differences between the three architectures remains quite constant. This was expected since a major part of the overall area is occupied by the RAM. Other differences between the three designs with regard to their logic components are almost negligible. Table 1 lists an area breakdown of the ZORRO ASIC after synthesis for 5 ns. It shows that our *3-Share*, *3-Share**, and *4-Share* architectures require 13.4 kGE, 13.9 kGE, and 16.2 kGE, respectively. Table 2 lists a comparison between ZORRO and related KECCAK-based ASIC designs in the field of low-area and DPA-security.

For the actual tapeout-version of ZORRO, we added a couple of DFT circuitries in order to provide suitable testing possibilities. This, the insertion of the required buffers, and the fact that after the backend design a realistic wireload model was available, lead to an increase in area to 14 kGE, 14.5 kGE, and 17 kGE for the *3-Share*, *3-Share**, and *4-Share* architectures, respectively. Fig. 7 shows the final layout of ZORRO as well as a photo of the chip. Table 3 provides a datasheet for some of ZORRO's final specifications.

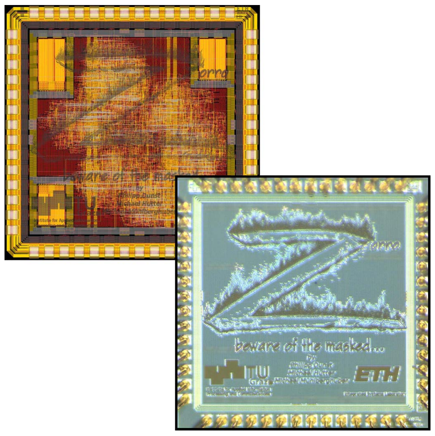


Fig. 7. Chip layout and photo of ZORRO.

Table 3. Datasheet of ZORRO.

Property	
Technology (UMC)	0.180 µm
Supply Volt. (Core/Pad)	1.8 V/3.3 V
Max. Frequency (f_{max})	200 MHz
Required Area	46.0 kGE
Est. Power Cons. @ f_{max}	
3-Share	17.3 mW
3-Share*	19.7 mW
4-Share	20.8 mW
Crypt. Perf. (Normal/Masked)[1]	
3-Share	21'888/113'184
3-Share*	21'888/113'184
4-Share	21'888/149'640

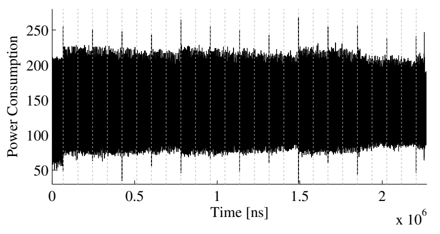
[1] Requ. cycles for one KECCAK-*f* perm.

4.2 Power-Analysis Results

In order to validate our design regarding power-analysis resistance, we performed power measurements and applied a standard Correlation Power Analysis (CPA) based on the Pearson correlation coefficient [10] on the measured power traces as a first step. Furthermore, higher-order CPA attacks were performed targeting the *3-Share* and *3-Share** implementations. For the rest of this section, ρ_c indicates the correlation coefficient of the correct key guess. Another procedure to rate the power-analysis resistance of an implementation is the method presented in [13] based on the statistical t-test. The advantage of the t-test is that no leakage model has to be defined. We have observed Hamming-distance leakage of intermediate values during simulation runs, so we decided to perform CPA attacks as a first step. For future work we will also investigate the t-test methodology and compare the outcome with the results presented in this work. Due to the time-consuming measurement process and the huge number of required measurements, we did not investigate the *4-Share* implementation so far.

We used a *Picoscope 6404c* oscilloscope to capture the power traces from the ZORRO ASIC. The voltage drop across a 1 Ω resistor in the core supply line was measured by applying a *LeCroy AP033* differential probe. This setup allows to minimize the noise created by, for instance, I/O activity of the chip because the chip has a separate supply line for the I/O part. The traces were recorded with a sampling rate of 1 GS/s and the clock frequency of the ASIC was set to 10 MHz.

The first power trace. The left plot in Fig. 8 shows a measured power trace of an entire KECCAK-*f* permutation of ZORRO running in *normal mode*. It shows all 24 rounds (including one additional round at the end where ρ is skipped) separated by a dotted vertical line. The right plot in Fig. 8 shows a zoom into the first round. We separated the slice and lane processing phases with a dashed vertical line as well as the eight slice-processing iterations (ZORRO processes the 64 slices in eight blocks) by dotted vertical lines. The same was done for the six



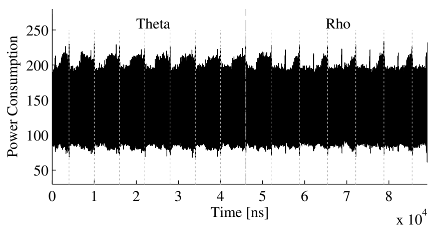


Fig. 8. Power trace of an entire KECCAK-f permutation while ZORRO is running in *normal mode* (left plot). Zoom into the first round, computing θ and ρ (right plot).

lane-processing iterations (ZORRO processes all 24 lanes in blocks of four). The time interval where the θ step of the first round takes place is of special interest because the power-analysis attacks presented in the following target the θ step. Only the first θ step was recorded for the power analysis attacks in order to keep the amount of data small.

Performing CPA. CPA attacks presented in this work focus on the first round of KECCAK-f. In particular, we targeted a storage operation of the 256-bit *Sub-State* register that stores key-dependent intermediate values during the θ step. The decision to target the θ transformation and not the non-linear χ transformation was motivated by the modified round schedule. In the first round, θ is the only slice-based transformation leading to a simple power model. We target a (unknown but constant) 256-bit key that gets concatenated with the (known and random) associated data. Thus, each targeted slice operation reveals information about four key bits. Since θ processes two slices in parallel, we can efficiently target 8 key bits by evaluating 256 key hypotheses. θ is a linear function, so not only the correct key will result in a high correlation, but shifted key variants will correlate too. Therefore, this attack does not only reveal the correct key but it will also reveal a small set of other possible key candidates (in our experiment we will get up to eight out of 256 possible key candidates). Due to the fact that we can attack all 64 slices and four bits of the key of two subsequent slice pairs must be similar, only eight key candidates with a length of 256 bits remain for a brute-force attack, what is within computational bounds (e.g., when attacking the slice pair (1,2) followed by the slice pair (2,3), only key candidates where the key bits of slice two are similar need to be considered). First experiments showed that ZORRO leaks the information according to the Hamming distance power model [16]. As a reference, preliminary attacks target ZORRO running in *normal mode* (*NM*, no countermeasures enabled). After the initial attacks, we have activated the DPA countermeasures one after the other in order to evaluate their impact on the power-analysis attacks. We first evaluate the *hiding mode (HM)* followed by the *masked mode (MM).*

Table 4. Results for the power-analysis attacks on ZORRO running in *hiding mode (HM).*

Mode	ti	Windowing	$\rho_{c,theory}$	$\rho_{c,pract}$
HM 1	16	no	0.044	0.049
HM 1	16	yes	0.176	0.237
HM 2	24	no	0.029	0.031
HM 2	24	yes	0.152	0.160
HM 15	128	no	0.005	-
HM 15	128	yes	0.062	0.057

Table 5. Min. number of measurements required.

Mode	N_{meas}
NM	<100
HM 1	285
HM 2	625
HM 15	4 925
HM 240†	70 000
MM 3-share	70 000

† not supported by ZORRO

Normal mode: The CPA attack was performed with 1 000 power traces leading to $\rho_c = 0.7$[6]. This ρ_c value indicates that less than 50 measurements are sufficient to distinguish the correct key hypothesis from the wrong key hypotheses [16].

Hiding mode: Next, we have activated hiding on the ZORRO ASIC. The *number of dummy rounds* (N_{dr}) has been set to 1 (*HM 1*), which means that zero or one dummy operation is randomly inserted in front of the first KECCAK-f permutation working on the real data. Moreover, as soon as the hiding mode is activated, the execution order during each slice-processing operation is randomized. As a result, the targeted operation can appear at 16 different *time instances ti*. According to [16], ρ_c should decrease by a factor of $\frac{1}{ti}$ compared to the unprotected case. When taking into account ρ_c of 0.7 for the unprotected case and $ti = 16$, this leads to an expected $\rho_{c,theory}$ value for the protected implementation of $\frac{0.7}{16} = 0.044$. Attacks on the protected implementation yield $\rho_{c,pract} = 0.049$, what fits well with theory. Next, windowing has been applied combining all the 16 time instances. According to [16], windowing should increase ρ_c by a factor of $\sqrt{16}$ for our attack. With windowing applied, our practical attacks yield $\rho_{c,pract} = 0.237$, what is significantly higher than the expected value $\rho_{c,theory} = 0.176$. Further practical experiments have been performed with $N_{dr} = 2$ (*HM 2*) leading to a $\rho_{c,pract}$ value of 0.031 without and 0.160 with windowing applied. Again, these values fit well with theory. ZORRO allows a maximum N_{dr} of 15 (*HM 15*), here practical results with windowing applied yield $\rho_{c,pract} = 0.057$ what is again close to $\rho_{c,theory} = 0.062$. Without windowing applied, no significant results can be observed with 100 000 measurements. Due to that reason, for *HM 15* without windowing we can only give $\rho_{c,theory} = 0.005$. Table 4 summarizes the results with regard to the hiding mode.

Masked mode: In a next experiment we performed power-analysis attacks targeting the first θ step on ZORRO running in *masked mode* (hiding was deactivated for this experiment). 1st-order CPA attacks using 100 000 power traces captured from the ASIC did not succeed. No significant correlation peaks could

[6] The confidence interval of the coefficient, where 99.99 % of all samples (4-σ border) are located in the normal distribution model for 1 000 traces, is about 0.12.

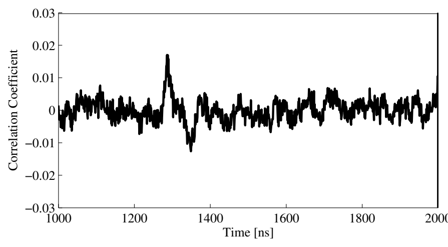


Fig. 9. 3rd-order CPA result for the correct key guess using 100 000 ASIC traces (Zorro running in *masked mode*).

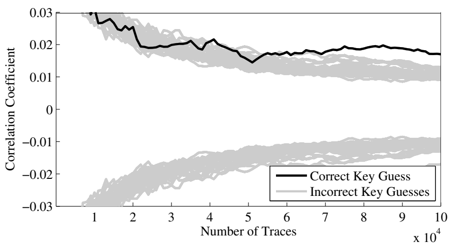


Fig. 10. Course of the correlation coefficient of Zorro running in *masked mode* (3rd-order CPA).

be observed in the result. Due to the clear patterns in the power traces, the time instances, where the first θ steps of each share are performed, can be identified with small effort. By combining the revealed time instances, a 3rd-order CPA attack has been mounted. The centralized product combining has been used as combination function, as suggested by Prouff et al. [21]. As shown in Fig. 9, this attack results in a significant correlation peak for the correct key hypothesis with $\rho_c = 0.016$. Fig. 10 shows the course of the correlation coefficients for all key guesses. With less than 70 000 measurements the correct key hypothesis can be distinguished from the wrong key hypotheses. Note that since the modifications between the *3-Share* and *3-Share** implementation solely affect the χ step (and not the herein targeted θ operation), the results of the 3rd-order CPA are identical for both three-share based architectures.

Comparison masking mode and hiding mode: Our first attack results show that both hiding as well as masking increase the effort for an attack. Attacks on the implementation using hiding also succeed without any modification of the traces (e.g., windowing). But, if windowing is applied, ρ_c only decreases by a factor of $\frac{1}{\sqrt{ti}}$ instead of $\frac{1}{ti}$. That means, windowing drastically reduces the number of required measurements N_{meas} for performing a successful attack. Attacks on the masked implementation do not succeed without combination of the traces, at least not if the shares are calculated sequentially during the first θ step, as it is the case with Zorro. In order to reach the same security level with hiding as with masking, 240 dummy rounds would be required. Each additional dummy round leads to eight additional time instances for the targeted operation to appear, so $ti = 240 \cdot 8 = 1920$ for 240 dummy rounds. As a consequence, ρ_c decreases to $\frac{0.7}{\sqrt{1920}} = 0.016$. This is now equal to ρ_c achieved with a 3rd-order CPA targeting the masked implementation. However, with 240 dummy rounds, the runtime of the implementation running in *hiding mode* exceeds the runtime of the implementation running in *masked mode*. Table 5 summarizes the number of required measurements N_{meas} for a successful key recovery. For the unprotected case (*NM*), less than 100 measurements are sufficient, for the hiding mode with the weakest protection (*HM1*), $N_{meas} = 285$, and for the hiding mode with the highest protection (*HM15*), $N_{meas} = 4\,925$ respectively. Note that N_{meas}

for *HM 1*, *HM 2*, and *HM 15* all assume that windowing is applied. Successful attacks targeting the masked mode (*MM 3-share*) require 70 000 measurements. 240 dummy rounds (*HM 240*) would also yield $N_{meas} = 70\,000$ but this mode is not supported by Zorro since 15 is the maximum number of dummy rounds.

5 Conclusions and Future Work

Zorro represents the first actually taped-out ASIC, hosting Keccak-based authenticated encryption systems secured against DPA attacks. It contains three distinct architectures, which solely differ with regard to the implemented secret-sharing technique. In addition to the DPA-secure designs, we aimed at low-area hardware architectures, targeting resource-constrained applications.

As future work, we are looking forward to investigate more DPA attacks targeting different intermediates of Keccak. In addition to target the output of the θ step, we want to target the output of the χ step as effects like glitches or early-propagation might allow successful DPA attacks with a lower order than 3. This will also enable a comparison of the three masking-secured implementations. Furthermore, we want to increase the number of traces up to several millions to identify unintended leaks. Finally, we plan to apply powerful Test Vector Leakage Assessment (TVLA) tests (including the fixed-vs-random t-test) to detect non-specific leakages and to verify the DPA resistance of our cores.

Acknowledgements. This work has been supported in part by the Swiss Commission for Technology and Innovation (CTI) under project number 13044.1 PFES-ES and in part by the European Commission through the FP7 program under project number 610436 (project MATTHEW) and by the Austrian Research Promotion Agency (FFG) and the Styrian Business Promotion Agency (SFG) under grant number 836628 (SeCoS). Moreover, we would like to thank the people from the Microelectronics Design Center of ETH Zurich for their support during the backend design of the ASIC. We also want to thank Svetla Nikova and Joan Daemen for their comments on the pre-print version of the paper.

References

1. CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness. `http://competitions.cr.yp.to/caesar.html`, March 2013.
2. G. Bertoni, J. Daemen, N. Debande, T.-H. Le, M. Peeters, and G. Van Assche. Power Analysis of Hardware Implementations Protected with Secret Sharing. Cryptology ePrint Archive: Report 2013/067, February 2013.
3. G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche. Duplexing the Sponge: Single-Pass Authenticated Encryption and Other Applications. In *SAC 2011*, volume 7118 of *LNCS*, pages 320–337. Springer, 2011.
4. G. Bertoni, J. Daemen, M. Peeters, G. V. Assche, and R. V. Keer. Keccak Implementation Overview, May 2012. `http://keccak.noekeon.org/Keccak-implementation-3.2.pdf(Version3.2)`.

5. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Sponge functions. ECRYPT Hash Workshop, Barcelona, Spain, May 24-25, 2007. `http://sponge.noekeon.org/SpongeFunctions.pdf`.
6. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. KECCAK specifications, Version 2 – September 10, 2009. `http://keccak.noekeon.org/Keccak-specifications-2.pdf`, September 2009.
7. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Building Power Analysis Resistant Implementations of Keccak. In *2nd SHA-3 Candidate Conference*, 2010.
8. B. Bilgin, S. Nikova, V. Rijmen, V. Nikov, J. Daemen, and G. V. Assche. Efficient and First-Order DPA Resistant Implementations of KECCAK. In A. Francillon and P. Rohatgi, editors, *CARDIS 2013*, volume 8419 of *LNCS*. Springer, 2013.
9. N. Borisov, I. Goldberg, and D. Wagner. Intercepting Mobile Communications: The Insecurity of 802.11. In M. Naghshineh and M. Zorzi, editors, *MobiCom 2001*, pages 180–189. ACM, 2001.
10. E. Brier, C. Clavier, and F. Olivier. Correlation Power Analysis with a Leakage Model. In M. Joye and J.-J. Quisquater, editors, *CHES 2004*, volume 3156 of *LNCS*, pages 16–29. Springer, 2004.
11. B. Canvel, A. P. Hiltgen, S. Vaudenay, and M. Vuagnoux. Password Interception in a SSL/TLS Channel. In *CRYPTO*, volume 2729 of *LNCS*, pages 583–599, 2003.
12. B. Gérard, V. Grosso, M. Naya-Plasencia, and F.-X. Standaert. Block Ciphers That Are Easier to Mask: How Far Can We Go? In *CHES 2013, Santa Barbara, CA, USA, August 20-23*, volume 8086 of *LNCS*, pages 383–399, 2013.
13. B. J. Gilbert Goodwill, J. Jaffe, and P. Rohatgi. A testing methodology for side-channel resistance validation. In *NIST Non-invasive attack testing workshop*, 2011.
14. E. B. Kavun and T. Yalcin. A Lightweight Implementation of Keccak Hash Function for Radio-Frequency Identification Applications. In S. B. O. Yalcin, editor, *RFIDsec*, volume 6370 of *LNCS*, pages 258–269. Springer, 2010.
15. P. C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In *CRYPTO 1999*, pages 388–397, 1999.
16. S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks – Revealing the Secrets of Smart Cards*. Springer, 2007. ISBN 978-0-387-30857-9.
17. S. Nikova, V. Rijmen, and M. Schläffer. Secure Hardware Implementation of Non-linear Functions in the Presence of Glitches. In *ICISC 2008*, pages 218–234, 2008.
18. S. Nikova, V. Rijmen, and M. Schläffer. Secure Hardware Implementation of Non-linear Functions in the Presence of Glitches. *Journal of Cryptology*, 24(2):292–321, 2011.
19. NIST. *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions (DRAFT FIPS PUB 202)*, May 2014.
20. P. Pessl and M. Hutter. Pushing the Limits of SHA-3 Hardware Implementations to Fit on RFID. In *CHES 2013*, volume 8086, pages 126–141. Springer, 2013.
21. E. Prouff, M. Rivain, and R. Bévan. Statistical analysis of second order differential power analysis. *Computers, IEEE Transactions on*, 58(6):799–811, 2009.