Side Channel Analysis of the SHA-3 Finalists

Michael Zohner [1,3], Michael Kasper [2,3], and Marc Stöttinger [1,3]

1: Technische Universität Darmstadt, Integrated Circuits and Systems Lab, Hochschulstraße 10, 64289 Darmstadt, Germany
e-mail: zohner,stoettinger@iss.tu-darmstadt.de
2: Fraunhofer Institute for Secure Information Technology (SIT), Rheinstraße 75, 64295 Darmstadt, Germany
e-mail: michael.kasper@sit.fraunhofer.de
3: Center for Advanced Security Research Darmstadt (CASED), Mornewegstraße 32, 64289 Darmstadt, Germany
e-mail: {michael.zohner,michael.kasper,marc.stoettinger}@cased.de

Abstract. At the cutting edge of today's security research and development, the SHA-3 competition evaluates a new secure hashing standard in succession to SHA-2. The five remaining candidates of the SHA-3 competition are BLAKE, Grøstl, JH, Keccak, and Skein. While the main research focus has been on the algorithmic security of the candidates, a side channel analysis has only been performed for BLAKE and Grøstl [4]. In this contribution we identify side channel vulnerabilities for JH-MAC, Keccak-MAC, and Skein-MAC and demonstrate attacks on their respective reference implementation. Additionally, we revisit the side channel analysis of Grøstl and introduce a side channel attack, which is able to recover the input to the hash function using only the measured power consumption and thus emphasizes the importance of side channel resistant implementations of hash functions.

1 Introduction

Hash functions are one of the elementary and most well established primitives in cryptography. Hash functions not only ensure the integrity of a transferred document, they are also applicable for functionalities including digital signatures, password verification, zero knowledge proofs, and *Message Authentication Codes (MACs)*. Thus, hash functions are part of the foundation that secures information technology. To provide hash functions that guarantee the required security features the National Institute of Standards and Technology (NIST) specified the *secure hashing algorithm* 1 and 2 (SHA-1 and SHA-2). When doubts about the security of SHA-1 and SHA-2 were raised, NIST announced the SHA-3 competition, which evaluates a successing hashing standard. In the current final round, the remaining candidates of the SHA-3 competition are *BLAKE*, *Grøstl*, *JH*, *Keccak*, and *Skein*. The algorithmic security of these candidates has already been researched down to the core and very few weaknesses have been found [24]. However, because the new SHA-3 standard will be implemented in various hardware architectures, its resistance against implementation attacks is also of great concern [16]. Such implementation attacks are for instance *side channel attacks*,

which utilize all kinds of physical leaking information, e.g. the execution time of the algorithm, the power consumption of the device, or even the electromagnetic emission, in order to recover secret information. Until now only the side channel resistance of BLAKE and Grøstl has been analyzed [4]. The side channel resistance of the other three candidates still remains uncertain and thus poses a potential risk for hardware implementations.

In this paper, we continue the work of Benoît et al. [4] by performing a side channel analysis for the three SHA-3 candidates JH, Keccak, and Skein. Subsequently, we present attacks that exploit the identified vulnerabilities and apply them to the respective reference implementation of the candidates, executing their dedicated MAC function. Furthermore, we demonstrate how a profiling based side channel attack can be used in order to recover the input to a Grøstl hashing operation. Note that the attacks, presented in this contribution, target the respective reference implementation of the candidates. An attack on a real world implementation of the candidates would (probably) be more challenging. However, the aim of this contribution is to identify and highlight side channel vulnerabilities of the candidates such that efficient countermeasures can be developed.

2 Background

The following section will briefly cover the background theory required for understanding this work. First we give an overview of hash functions, followed by a brief introduction to side channel analysis.

2.1 Hash functions

Hash functions $H : \{0,1\}^* \mapsto \{0,1\}^n$ map a variable sized data value from an input domain to a fixed sized representation from their output range, the so called hash. When used in cryptography, hash functions have to guarantee certain properties, e.g. collision resistance, second preimage resistance, and one-wayness. In order to simplify the development of hash functions so called *constructions* were proposed. Most prominent is the *Merkle-Damgård* construction, which allows building collision resistant hash functions from collision resistant compression functions. A hash function, built by the Merkle-Damgård construction, splits the message M into smaller blocks $M = (m_0, m_1, ..., m_{p-1})$ and iteratively processes them by calling the underlying compression function G and connecting the compression function calls by passing a *state value* H_i (c.f. Figure 1). The size of the state value is referred to as the *state size* of the hash function. A collision resistant compression function, which is required by the Merkle-Damgård construction, can again be built by applying the Matyas-Meyer-Oseas construction to a symmetric block cipher. Preneel et al. [21] performed a thorough analysis of 64 basic schemes for constructing compression functions from block ciphers, so called PGV schemes. Out of the 64 PGV schemes, Preneel et al. deemed twelve schemes secure. The first of these twelve secure schemes is the same as the

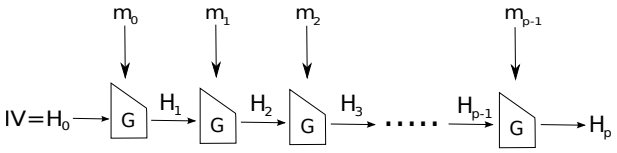


Fig. 1: Merkle-Damgård Construction

Matyas-Meyer-Oseas construction and similar to *unique block iteration (UBI)*, a construction that the SHA-3 candidate Skein is built on.

Another proposal for building hash functions is the sponge construction [5], which hashes a value by iteratively calling a permutation. The sponge construction divides the state value into the *bitrate* and *capacity* and digests a message by XORing it with the bitrate and then inserting the resulting state value into the permutation.

An application of hash functions is to compute MACs, which are used to authenticate data in cryptography. A MAC is computed by hashing a message together with a secret, which is only known to the sender and the receiver. The most prominent MAC function is *Hash-based MAC (HMAC)* [18]. HMAC computes a MAC as:

$$HMAC(K, M) = \mathcal{H}((K \oplus OPAD) \mid\mid \mathcal{H}\ ((K \oplus IPAD) \mid\mid M)), \tag{1}$$

where M is the message, K the key, $\mathcal{H}$ the underlying hash function, and $IPAD$ and $OPAD$ are two constants defined as the hexadecimal sequence $(3636...36)_{16}$ and $(5C5C...5C)_{16}$ with the same length as the state size of $\mathcal{H}$. Another MAC function is the envelope MAC ENV_{MAC} [12], which is defined as:

$$ENV_{MAC}(K, M) = \mathcal{H}(\overline{K}||\overline{M}||K) \tag{2}$$

where K is the key, $\overline{K}$ is the key padded to the state size of $\mathcal{H}$, and $\overline{M}$ is the message padded to a multiple of the state size.

2.2 Side Channel Attacks

Side channel attacks target cryptographic implementations and exploit all kinds of unintentionally emitted information, which can be attained during the computation of an algorithm. Such emitted information are for instance power consumption, electromagnetic emanations, or timings and micro-architectural characteristics. Side channel attacks can be divided into different classes, depending on the information they utilize in order to recover the key. *Power attacks*, for example, are based on the fact that a dependency between the power consumption of a device and the processed data exists [19]. An attacker can utilize and exploit this dependency in order to recover secret information, such as keys of cryptographic algorithms.

One of the most common power attacks is the *Differential Power Analysis (DPA)*. The DPA computes hypothetical power consumptions for each input

and key and compares them to the recorded power consumption (so called *power trace*) of the device. The hypothetical power consumption is computed by using a leakage model, i.e. the *Hamming weight (HW)*, that models the power consumption of the device and a hypothesis function that predicts a processed intermediate value. Finding a suited intermediate value, which reveals information about the key, is specified as *leakage analysis*. In order to compare the calculated hypotheses to the measured power consumption methods like the Pearson correlation or the difference in means can be used [19].

Profiling based attacks are another kind of power attacks that require a more powerful attacker model but are more effective concerning the number of attack traces. Profiling based power attacks are divided in two phases. First, a profiling phase builds a power consumption model from the recorded power intake of a training device. Secondly, an attacking phase recovers the processed data by measuring the power consumption of the actual target device and comparing it to the power consumption models, built in the profiling phase.

Another method for performing a profiling based attack is the machine learning algorithm *Support Vector Machines (SVM)*. SVM assigns a class to an input sample by using training data to construct a hyperplane in a higher dimensional space, which separates two classes of data [9]. Hospodar et al. [15] examined the applicability of SVM when analyzing power traces by evaluating its accuracy on several use cases and comparing it to the accuracy of a template attack. For the defined use cases, profiling with SVM performed similar to profiling with a template attack, and thus Hospodar et al. deemed revealing cryptographic keys from power traces, using SVM, feasible.

The classical template attack builds a power consumption template for every intermediate value. Thus, the complexity of the profiling phase is very high. In order to counter this problem, the intermediate values can be divided into classes that are characteristic for the leakage model (i.e. the Hamming weight) and a template can then be built for each class. However, since now a set of values is predicted, additional methods are needed to reveal the actual processed value. Therefore, profiling based power attacks were combined with *algebraic side channel attacks* [22], which build a system of equations that describe the algorithm and reveal the actual value when solved.

3 Side Channel Attacks on Hash Functions used as HMAC

3.1 Side Channel Attacks Against HMACs Based on PGV Schemes

Okeya [20] performed a side channel analysis of the twelve PGV schemes when computing a HMAC. The side channel analysis indicated that the DPA could reveal the information, required to forge a HMAC, for the eleven of the twelve PGV schemes, even if the underlying block cipher was resistant to side channel attacks. The information, required for forging a HMAC, are the *inner keyed state* and the *outer keyed state*, i.e. the state value resulting from the digestion of the

key XORed with the inner pad and outer pad. Only the first PGV scheme, which is the same as the Matyas-Meyer-Oseas construction and UBI, was deemed side channel resistant. In response to the SHA-3 competition, Okeya et al. [13] also presented a side channel analysis of several MAC functions.

3.2 Side Channel Analysis of Six SHA-3 Candidates

Benoît et al. [4] presented a side channel analysis of six round two SHA-3 candidates, computing a HMAC, and outlined vulnerabilities to which the DPA could be applied. Among the six analyzed candidates were the two finalists BLAKE and Grøstl [3] [12]. For BLAKE-HMAC it was shown that the DPA could recover the inner keyed state and the outer keyed state by exploiting the leakage of the modular addition and the XOR. For Grøstl-HMAC the XOR and the S-box were identified as exploitable operations when recovering the keyed state. However, the authors recommended the exploitation of the S-box since their evaluation function yielded better results for the S-box than for the XOR.

4 Analysis of the SHA-3 Candidates

We conducted a leakage analysis of the SHA-3 candidates and were able to identify side channel vulnerabilities. Following, we give an overview of our analysis on four out of five finalists of the SHA-3 competition and sketch the respective attacks. The remaining candidate, BLAKE, has already been analyzed by Benoît et al. [4] and will therefore be left out of scope. The attacks on the candidates were conducted on an ATMega-256-1 microcontroller with a register size of 8 bit, which was measured using a PicoScope 6000 at a frequency of 8 MHz.

In order to explain the attacks, we will state the hypothesis functions, used in the conducted DPAs. Note, that this hypothesis functions are tailored to the ATMega-256-1 microcontroller and are not universally applicable. However, the generalized attack principle can be adopted in order to fit other target devices. The hypotheses and the measured power traces were compared using the Pearson correlation. To identify the point in time, at which the power consumption of the operation occurs, we varied the area for which we computed the Pearson correlation until for the correlation converged for a hypothesis. In total we required 200 power traces to successfully recover the processed key for the attack against JH-MAC, Keccak-MAC, and Skein-MAC. For the profiling based attack against Grøstl we measured 4500 power traces in the profiling phase and one power trace in the attack phase.

In order to illustrate our practical results, we added example correlations of each attack as well as a picture of a SVM classifier in Appendix B.

4.1 Skein

Description: Schneier et al. [11] proposed the hash function Skein, which is built by applying UBI, to the block cipher *Threefish*. Threefish is a block cipher with three different internal state sizes, i.e. 256 bit, 512 bit, and 1024 bit.

Threefish processes the input message M and key K by dividing them into N 64 bit blocks (with $N \in \{4, 8, 16\}$) $M_0, M_1, ..., M_{N-1}$ and $K_0, K_1, ..., K_{N-1}$ and performing a modular addition:

$$H_i = (M_i + K_i) \bmod 2^{64}, \text{ for } 0 \leq i < N. \tag{3}$$

Subsequently, H_i is processed using the operations MIX, Permute, and AddRoundKey.

Skein provides its own MAC functionality by padding the key to a multiple of the state size and digesting it before the message (c.f. Figure 2). Therefore, the *keyed state*, i.e. the state value resulting from the digestion of the key, is constant for a fixed key. Thus, in order to forge a legitimate Skein-MAC we either need the key or the keyed state value.

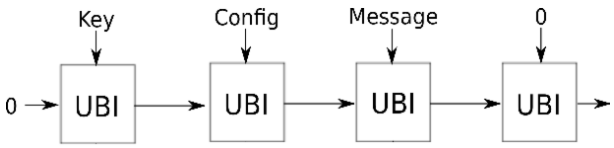


Fig. 2: Skein-MAC

Side channel analysis: Since the power consumption during the digestion of the key is always constant for a fixed key and therefore not attackable using the DPA, we targeted the keyed state. Also, because Skein uses UBI, which is side channel secure (see Okeya [20]), we attacked Threefish by performing the DPA on the result of the modular addition between the keyed state and the input message [17].

The problem with the DPA on Skein-MAC is that the modular addition processes two 64 bit values. This renders the usual approach of performing the DPA, by computing hypotheses for all 2^{64} key candidates, computationally infeasible. Thus, in order to make a DPA on Skein feasible, the attacked 64 bits of the keyed state value were split into eight blocks of eight bit each and attacked independently[1]. The hypothesis function h for the DPA is:

$$h(m_j)_c = HW(m_j + c), \text{ for } 0 \leq j < 8, \tag{4}$$

where $m_0 m_1 ... m_7$ is the 64 bit block input message and $c \in \{0, 1, ..., 255\}$ denotes the key hypothesis. This attack has to be performed for each of the $N \in \{4, 8, 16\}$ 64 bit blocks of the keyed state value. But since the N 64 bit blocks are processed independently, the complexity for attacking multiple blocks rises only linearly when attacking Skein versions with a larger state size.

[1] The size and number of the sub-blocks was chosen as a trade-off between the complexity of computing the hypotheses and the noise level due to omitted bits for the attacked 8-bit architecture.

4.2 JH

Description: Hongjun Wu proposed the hash function JH [25], designed by using a novel construction method for building a collision resistant compression function from a block cipher and he also introduced a novel underlying block cipher E, built by generalizing the AES design methology. The JH construction processes the 512 bit message M_i and the 1024 bit state value H_i as follows:

$$H_{i+1} = E(H_i \oplus (M_i \mid\mid \{0\}^{512})) \oplus (\{0\}^{512} \mid\mid M_i). \tag{5}$$

Upon being called, E changes the order of the bits of the input H. This transformation is called *grouping* and is illustrated in Figure 3. The result of the grouping is a state with 256 four bit blocks. Subsequently, the four bit blocks are substituted by applying two 4 bit S-boxes S_0 and S_1. Which S-box is used for which four bit block is determined by predefined round constants. The suggested MAC function for JH is HMAC.

Side channel analysis: The following attack is applicable for the recovery of the inner and outer keyed state of JH-HMAC, thus we will describe it in a general manner. In order to recover the 1024 bit sized state value H_K, resulting from the digestion of the key K, the leakage of two operations has to be exploited. The first exploited operation is the XOR of JH's construction, which processes the message M and $H_{K,0}$, i.e. the first 512 bits of $H_K = H_{K,0} \mid\mid H_{K,1}$. Because $H_{K,0}$ is directly XORed with the input message M and the XOR is an operation, which is known to be exploitable by the DPA, the hypothesis function h_0 was chosen as:

$$h_0(M)_c = HW(M \oplus c), \text{ for } c \in \{0,1\}^{512}. \tag{6}$$

Similar to the attack against Skein, the hypothesis computations are divided in blocks of eight bit size in order to reduce the complexity of the hypothesis computation.

After applying the attack on the XOR of the construction, we have recovered the first 512 bit ($H_{K,0}$) of the state value H_K, but still require the last 512 bits ($H_{K,1}$) in order to forge legitimate JH-HMACs. Thus, we have to perform a second attack, which targets the S-box operation during the execution of the block cipher.

After the grouping of the block cipher, the state $A = (a_0, a_1, ..., a_{255})$ consists of 256 four bit blocks, which are substituted using the two 4 bit S-boxes. Since the grouping mixes the bits from $H_{K,0}$ and $H_{K,1}$, we know for each $a_i = (a_{i,0}, a_{i,1}, a_{i,2}, a_{i,3})$ the two leading bits $a_{i,0}$ and $a_{i,1}$ and our target is to recover the two trailing bits $a_{i,2}$ and $a_{i,3}$.

The hypothesis function h_1 for the second DPA is:

$$h_1(a_{i,0}, a_{i,1})_c = HW(S_{C^0(i)}(a_{i,0} \mid\mid a_{i,1} \mid\mid c)), \tag{7}$$

where $a_{i,j} \in \{0,1\}$, $C^0(i)$ is the round constant for the first round at position i, and $c \in \{0,1,2,3\}$ represents the two trailing bits. The attack is visualized

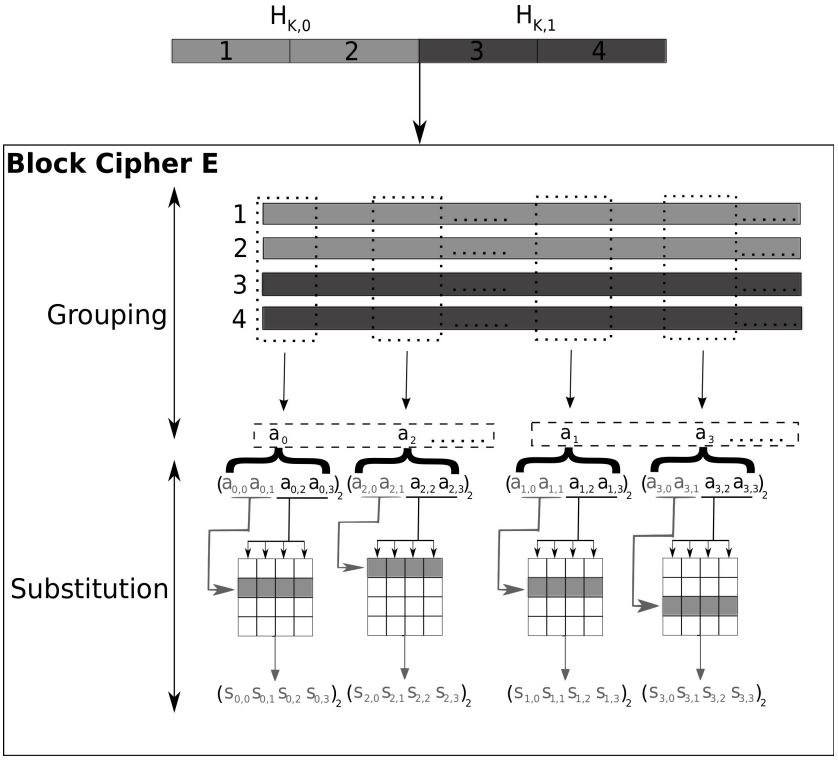


Fig. 3: Side Channel Attack against JH where the green values represent the known information.

in Figure 3, where the grey values represent the known and the black values the unknown information. Note, that in this specific case, we can only vary over four possible outputs instead of 16 for each a_i, since the two trailing bits are constant. This makes the DPA more complex, since different key candidates result in similar hypotheses for each output.

4.3 Keccak

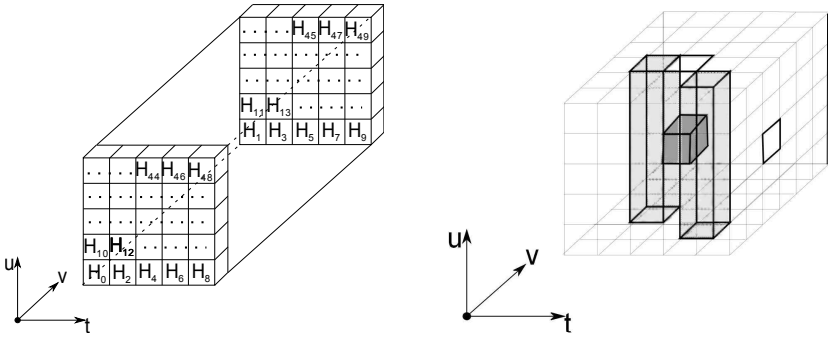
Description: Daemen et. al [6] proposed a family of permutation functions called Keccak, which are used by the sponge construction in order to build the corresponding hash function family. The Keccak permutation family members are denoted by $f[b]$, for $b = 25 \cdot 2^{\lambda}$ and $0 \leq \lambda < 7$, where b is the state size of the member λ. Before executing the permutation, Keccak transforms the input $H = H_0 H_1 ... H_{b-1}$, with $H_i \in \{0, 1\}$, into a three dimensional state matrix $A_{t,u,v}$ by computing:

$$A_{t,u,v} = H_{2^{\lambda} \cdot (5u+t)+v}, \text{ for } 0 \leq t, u < 5, \text{ and } 0 \leq v < 2^{\lambda}, \tag{8}$$

(c.f. Figure 4a). Keccak then performs $12+2\cdot\lambda$ rounds of an internal permutation R, which again consists of the five permutations: θ, ρ, ϕ, χ, and ι. In this paper we only focus on θ, because in terms of power attacks it reveals the most information. The permutation θ is defined as:

$$\theta : A'_{t,u,v} = A_{t,u,v} \oplus \bigoplus_{i=0}^{4} A_{t-1,i,v} \oplus \bigoplus_{i=0}^{4} A_{t+1,i,v-1}. \tag{9}$$

Figure 4b visualizes the permutation θ. Roughly speaking, θ first XORes all elements of the same column and then XORes two of these values again with each matrix element. Keccak-MAC is computed by hashing $(K||M)$.



(a) Keccak Assignment (b) Permutation θ

Fig. 4: Keccak Operations

Side channel analysis: Performing a DPA on Keccak-MAC is challenging, since the key is not padded. In theory, we have to distinguish between many possible cases, depending on the length of the key and the internal state size of Keccak. However, as a full analysis of all cases could fill a whole paper by itself, we will only introduce the main attack principle and mention further extensions of the attack. The presented attack is divided into two steps. The first step aims at recovering the bitrate of the sponge function by exploiting the XOR with the input. The second step then uses the knowledge of the first step to target the XOR, performed during the θ permutation.

Since the key is not padded when computing Keccak-MAC, the key as well as its size are unknown. Therefore, we do not know the number of message bits, which are digested in the same permutation iteration as the keyed state. However, the recovery and numbering can be done by performing the DPA starting with the first message bit over a varying area and repeatingly attacking the next message bit until no correlation in the examined area can be found (see Algorithm 1 in Appendix A). The reason for this procedure is that if two consecutive message bits are processed in the same permutation iteration, they will be processed successively by the XOR of the sponge construction, which can be verified

by analyzing the correlation, resulting from the DPA. If, on the other hand, they are processed in different permutation iterations, the permutation will separate the two operations and the point in time, where the highest correlation occurs, will differ strongly. As hypothesis function h_0 for the DPA we compute:

$$h_0(M_i)_c = HW(M_i \oplus c), \tag{10}$$

where $M_i \in \{0,1\}$ is the i-th message bit, processed in the first iteration, and $c \in \{0,1\}$. By performing this attack, we can recover the part of the bitrate, which is XORed with the first message bits.

The second step aims at recovering the remaining state value, i.e. the capacity and the part of the bitrate, which is processed with the key. When the state value is fed into the Keccak permutation, it is transformed into the state matrix representation and the permutation θ is performed. During θ, the reference implementation of Keccak precomputes the XOR of all rows by XORing all column elements (see Algorithm 2 in Appendix A). Roughly speaking, the precomputation XORes all five columns, starting with the two columns at the bottom rows, then the column in the 3rd row and so on it reaches the topmost column. Thus, due to the processing order of the precomputation, we can recover all elements of the state matrix, which are above a known column. The hypothesis function h_1 is:

$$h_1(A_{i,j})_c = HW(A_{i,j} \oplus c), \tag{11}$$

where $A_{i,j} \in \{0,1\}^\lambda$, $0 \leq i,j < 5$, and $c \in \{0,1\}^\lambda$. When the overlying elements are recovered, the attack is repeated until the complete state matrix, i.e. the keyed state value, is known.

This attack becomes challenging when the number of recovered bits of the bitrate is smaller than $\frac{b}{5}$, i.e. if we do not know all bits of at least one row. In this case, not every value can be recovered during the computation of θ and the attack has to be extended to the permutation χ and/or has to be conducted over multiple Keccak rounds. If, on the other hand, the key length is smaller than $\frac{b}{5}$ and if we know at least $\frac{b}{5}$ bits of the bitrate, the actual key can be recovered, since all key bits are directly processed with message bits during the precomputation of θ. Thus, while the missing padding of the key makes the attack more difficult, it also allows the recovery of the actual key if the key is chosen too small.

4.4 Grøstl

Description: Grøstl, proposed by Knudsen et al., is a Merkle-Damgård hash function, based on a variant of the block cipher standard AES [2][12]. Grøstl digests a message $M = M_0, M_1, ..., M_{N-1}$ by compressing M_i and the 512 (1024) bit state value H_i using a compression function f, defined as:

$$f(H_i, M_i) = H_{i+1} = P(H_i \oplus M_i) \oplus Q(M_i) \oplus H_i, \tag{12}$$

where P and Q are two permutations, similar to AES. P and Q both perform the operations AddRoundConstant, SubBytes, ShiftBytes, and MixBytes for 10

(14 if the state size is 1024 bit) rounds. The effect of these operations is the same as for AES, except that P and Q operate on a 8×8 (8×16) internal state matrix A, consisting of eight bit entries. Thus, the operations were modified in order to cope with the increased state size. The difference between P and Q are the round constants in the AddRoundConstant operation and the number of shifts in the ShiftBytes operation.

The SubBytes operation of P and Q is the same as for AES, i.e. it substitutes each element of A with the element in the corresponding Rjindael S-box. MixBytes, however, uses a different matrix B for multiplication due to the increased state size:

$$B_i = circ_i(02, 02, 03, 04, 05, 03, 05, 07), \text{ for } 0 \leq i < 8, \tag{13}$$

where B_i denotes the i-th row of B and $circ_i$ denotes the cyclic right shift by i positions. MixBytes performs the matrix multiplication as $A = B \times A$ in Rjindael's Galois field $GF(2^8)$ with the irreducible polynomial $x^8+x^4+x^3+x+1$.

Side channel analysis: Benoît et al. [4] proposed a DPA against Grøstl-HMAC, which targeted the inner - and outer keyed state. Also, a successful DPA on Grøstl-MAC, i.e. Grøstl computing the envelope MAC, with the attack strategy that Benoît et al. proposed, results in a key recovery. When attacking Grøstl-MAC, the recovery of the first padded key is identical to the recovery of the inner keyed state of Grøstl-HMAC. However, instead of subsequently processing the outer keyed state, Grøstl-MAC directly processes the actual key, allowing a key recovery if the DPA is successful.

Still, due to the vulnerability of AES to algebraic side channel attacks, we saw the necessity for a more thorough leakage analysis. Thus, we varied the attacker model and assumed an attacker with arbitrary access to a profiling device and access to the target device that is restricted to measuring the power consumption of one execution. Note that the assumed attacker does not possess knowledge of the input or output of the hash function. The goal of this attacker is therefore to recover the input to the hash function call by analyzing the power trace using power consumption models built during a profiling phase. Since we are limited to one power trace in the attack phase, we set up a system of equations for P and Q, which set intermediate values, i.e. inputs and outputs of successive operations, in relation. In our explicit case, the attack consists of three steps: first we build profiles for all HW of each intermediate values, secondly we try to recover the HW of the intermediate value, and lastly we insert the recovered HW into our system of equations that yields the processed message when solved.

For the task of determining the HW of the intermediate values, we chose SVM [9] [10] because of its good results in [15]. Since there has not been an approach of classifying the HW using SVM, we varied possible input parameters until we found the set, which produced the highest accuracy. As input features we chose the three biggest points of a power trace of an operation as well as their sum. This setup yielded an accuracy of 95.1% on test data.

In order to exploit the recovered HW information, we set up a system of equations for P and Q. The equations in this system consist of the HW of the input as well as the HW of first two rounds of the SubBytes and MixBytes results for every state value. The SubBytes and MixBytes operations were chosen because they set the most constraints on the processed values and thus are the most efficient when determining the processed data. We were able to solve the system of equations and thus recover the processed data using 200 Hamming weights.

The attack was performed using the reference implementation of Grøstl-256 with a 8×8 state matrix. In total needed to profile nine HW for five different operations. We did this by using 100 traces for each HW, which resulted in $5 * 9 * 100 = 4500$ measurements for the profiling phase. During the attacking phase, one traces was recorded and input into the constructed SVMs in order to classify the Hamming weights of the profiled operations. These Hamming weights were inserted into the algebraic system, which was solved by verifying the conditions, imposed by the equations.

5 Conclusion

We presented a side channel analysis of four out of five finalists of the SHA-3 competition. Potential vulnerabilities of JH, Keccak, and Skein were revealed and attacks on their dedicated MAC function were mounted. In order to demonstrate the real world applicability of the attacks, they were conducted on an ATMega-256-1 platform and the complexity and gain of the attacks was evaluated.

Furthermore, we presented a profiling attack against Grøstl, using SVM to recover Hamming weights of intermediate values and an algebraic system to recover the processed data by inserting the Hamming weights. The significance of the attack lies in the recovery of the data, Grøstl hashes. Recovering the processed data is a great issue when a secret is hashed. Therefore, while side channel attacks are not as threatening to hash functions as they are to encryption functions, they should not be left out scope when evaluating a hash function.

Note that side channel attacks against the reference implementations of the SHA-3 candidates do not determine how hard it is in practice to successfully attack SHA-3 candidate implementations, since reference implementations are designed for understandability. Therefore, this paper does not conclude that certain candidates are harder to attack in practice than others. The intention of this paper is to identify the operations, which an attack would exploit and to outline the necessity of side channel resistant hash functions in order to further encourage the development of countermeasures against side channel attacks on SHA-3 candidates. Some research on countermeasures has already been conducted by Hoerder et al. [14], who performed an evaluation of hash functions, including Skein, BLAKE, and Keccak, on a power analysis resilient processor architecture. Also, for Keccak there have been some notes on side channel countermeasures [23] as well as proposals for masked Keccak implementations [7][1].

The winner of the SHA-3 competition and therefore the new SHA-3 standard will be announced in spring 2012. Whichever of the finalists may win will be adopted in many implementations and devices, ranging from small embedded systems to high performance computers. Thus, fast, flexible, and secure implementations of the winner will have to be provided in order to cover all application scenarios. The insight, gained from our evaluation, is that none of the analyzed candidates should be blindly deployed as MAC function (or in the case of Grøstl as hash function) in a security sensitive context, without implementing countermeasures or at least evaluating the side channel resistance of the implementation. Thus, NIST should contribute to the side channel security of the new SHA-3 standard by specifying multiple, flexibly applicable countermeasures, in order to secure implementations of SHA-3.

References

1. Alemneh, E.: Sharing nonlinear gates in the presence of glitches (August 2010), http://essay.utwente.nl/59599/, Master's Thesis
2. Announcing The Federal: Processing Standards Publication 197
3. Aumasson, J.P., Henzen, L., Meier, W., Phan, R.C.W.: SHA-3 proposal BLAKE. Submission to NIST (Round 3) (2010)
4. Benoît, O., Peyrin, T.: Side-channel analysis of six SHA-3 candidates. In: Proceedings of the 12th international conference on Cryptographic hardware and embedded systems. pp. 140–157. CHES'10, Springer-Verlag, Berlin, Heidelberg (2010)
5. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Cryptographic sponge functions. Submission to NIST (Round 3) (2011), http://sponge.noekeon.org/CSF-0.1.pdf
6. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: The Keccak reference. Submission to NIST (Round 3) (2011)
7. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Building power analysis resistant implementations of Keccak. In: Second SHA-3 Candidate Conference (August 23-24, 2010)
8. Brabanter, K.D., Karsmakers, P., Ojeda, F., Alzate, C., Brabanter, J.D., Pelckmans, K., Moor, B.D., Vandewalle, J., Suykens, J.: Ls-svmlab toolbox users's guide version 1.7 (2010), http://www.esat.kuleuven.be/sista/lssvmlab/
9. Burges, C.J.C.: A Tutorial on Support Vector Machines for Pattern Recognition. Data Min. Knowl. Discov. 2(2), 121–167 (1998)
10. Chang, C.C., Lin, C.J.: LIBSVM: A library for support vector machines. ACM Transactions on Intelligent Systems and Technology 2, 27:1–27:27 (2011), http://www.csie.ntu.edu.tw/ cjlin/libsvm
11. Ferguson, N., Lucks, S., Schneier, B., Whiting, D., Bellare, M., Kohno, T., Callas, J., Walker, J.: The Skein Hash Function Family. Submission to NIST (Round 3) (2010)
12. Gauravaram, P., Knudsen, L.R., Matusiewicz, K., Mendel, F., Rechberger, C., Schläffer, M., ren S. Thomsen, S.: Grøstl – a SHA-3 candidate. Submission to NIST (Round 3) (2011)
13. Gauravaram, P., Okeya, K.: Side Channel Analysis of Some Hash Based MACs: A Response to SHA-3 Requirements. In: Proceedings of the 5th international conference on Information, Communication and Signal Processing. pp. 111–127. ICICS (2008)

14. Hoerder, S., Wojcik, M., Tillich, S., Page, D.: An evaluation of hash functions on a power analysis resistant processor architecture. In: Workshop in Information Security Theory and Practice - WISTP 2011. pp. 160–174. Springer-Verlag LNCS 6633 (June 2011)
15. Hospodar, G., Mulder, E.D., Gierlichs, B., Verbauwhede, I., Vandewallei, J.: Least squares support vector machines for side-channel analysis. In: 2nd Workshop on Constructive Side-Channel Analysis and Secure Design, COSADE (2011)
16. Kelsey, J.: How to Choose SHA-3, http://www.lorentzcenter.nl/lc/web/2008/309/presentations/Kelsey.pdf
17. Kerstin Lemke, K.S., Paar, C.: Dpa on n-bit sized boolean and arithmetic operations and its application to idea, rc6 and the hmac-construction (August 2004), in proceedings of CHES 2004, Springer Berlin / Heidelberg
18. Krawczyk, H., Bellare, M., Canetti, R.: HMAC: Keyed-Hashing for Message Authentication (1997)
19. Mangard, S., Oswald, E., Popp, T.: Power analysis attacks - revealing the secrets of smart cards. Springer (2007)
20. Okeya, K.: Side Channel Attacks Against HMACs Based on Block-Cipher Based Hash Functions. In: Batten, L., Safavi-Naini, R. (eds.) Information Security and Privacy, Lecture Notes in Computer Science, vol. 4058, pp. 432–443. Springer Berlin / Heidelberg (2006)
21. Preneel, B., Govaerts, R., Vandewalle, J.: Hash functions based on block ciphers: a synthetic approach. In: Proceedings of the 13th annual international cryptology conference on Advances in cryptology. pp. 368–378. Springer-Verlag New York, Inc., New York, NY, USA (1994)
22. Renauld, M., Standaert, F.X., Veyrat-Charvillon, N.: Algebraic Side-Channel Attacks on the AES: Why Time also Matters in DPA. In: Proceedings of the 11th International Workshop on Cryptographic Hardware and Embedded Systems. pp. 97–111. CHES '09, Springer-Verlag, Berlin, Heidelberg (2009)
23. The Keccak Team: Note on side-channel attacks and their countermeasures, http://keccak.noekeon.org/NoteSideChannelAttacks.pdf
24. TU Graz: The SHA-3 Zoo, http://ehash.iaik.tugraz.at/wiki/The_SHA-3_Zoo
25. Wu, H.: The Hash Function JH. Submission to NIST (round 3) (2011)

A Algorithms

Algorithm 1 Recovering the number i of processed message bits and the bitrate

Require: message $m = m_0 m_1 ... m_{N-1}$ with $m_i \in \{0, 1\}$, powertrace *trace*

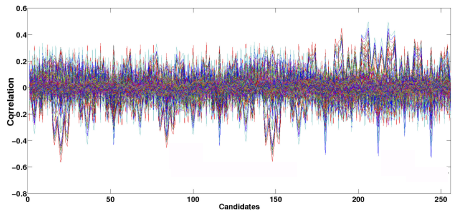
$i = 0$, $n = 1$;
repeat
  for j from 0 to 1 **do**
    $hypothesis[j] = HW(m_i \oplus j)$;
  end for
  $corr$ = Pearson-Correlation $(hypothesis, trace)$; i = i+1;
until *corr* does not converge anymore

B Correlations

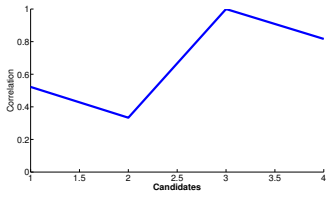
Algorithm 2 Precomputing the XOR over all columns in the permutation θ

Require: state matrix $A[t][u][v]$ with $t, u = 5$ and $v = 64$

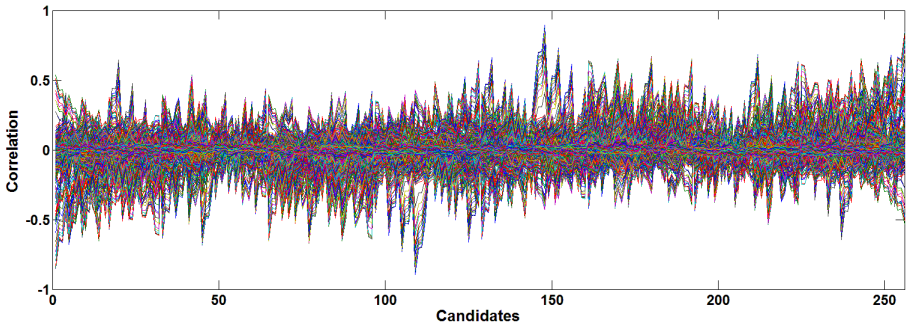
```
i ,j ,C[5];
for i from 0 to 5 do
    C[i] = 0;
    for j from 0 to 5 do
        C[i] = C[i] ⊕ A[i][j];
    end for
end for
```



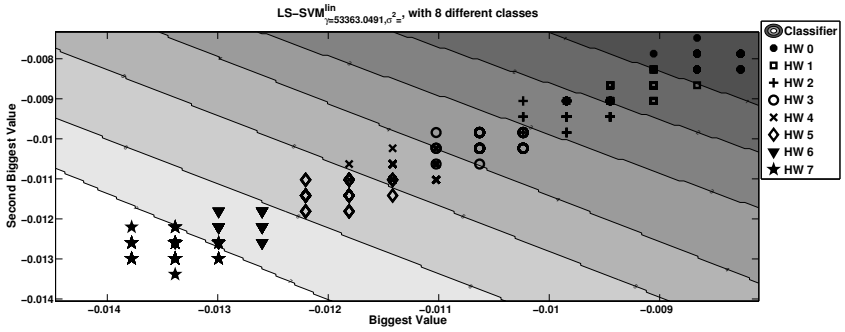
(a) Correlation Skein Modular Addition on 8 bit using 200 traces



(b) Correlation JH S-box S_0 using 200 traces (only 1 point in time)



(c) Correlation Keccak XOR on 8 bit using 200 traces



(d) Multiclass SVM separating 8 HW classes for Grøstl using [8]

Fig. 5: Results of the attack