Side Channel Analysis of the SHA-3 Finalists

Michael Zohner*, Michael Kasper†, Marc Stöttinger*, and Sorin A. Huss*

*Integrated Circuit and Systems Lab (ISS), Technische Universität Darmstadt,
Email: {zohner, stoettinger, huss}@iss.tu-darmstadt.de

†Fraunhofer Institute for Secure Information Technology (SIT)
Email: michael.kasper@sit.fraunhofer.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 focus was on the algorithmic security of the candidates, a side channel analysis has only been performed for BLAKE and Grøstl [1]. In order to equally evaluate all candidates, we identify side channel attacks on JH-MAC, Keccak-MAC, and Skein-MAC and demonstrate the applicability of the attacks by attacking their respective reference implementation. Additionally, we revisit the side channel analysis of Grøstl and introduce a profiling based side channel attack, which emphasizes the importance of side channel resistant hash functions by recovering the input to the hash function using only the measured power consumption.**

***Index Terms*—SHA-3 Finalists; Side-Channel Analysis; DPA**

I. 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 (MAC)*. Thus, hash functions are part of the foundation that secures information technology. To guarantee the security features of a hash function, the National Institute of Standards and Technology (NIST) published 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 major weaknesses have been found [2]. However, because the new SHA-3 will be implemented in various hardware architectures, its resistance against implementation attacks is also of great concern [3]. 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 [1]. The resistance of the other three candidates to side channel attacks still remains uncertain and thus poses a potential risk for hardware implementations.

In this paper, we continue the work of Benoît et al. [1] by performing a side channel analysis for the three SHA-3 candidates JH, Keccak, and Skein. Subsequently, attacks, based on the insights of the analysis, are introduced and applied to the respective reference implementation of the candidates, executing their dedicated MAC function. Furthermore, we perform a side channel analysis of Grøstl and introduce a novel power analysis attack against its hashing operation, which allows us to recover the input to the hash function call without any information about the message or hash.

978-3-9810801-8-6/DATE12/© 2012 EDAA

II. 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.

A. 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. The most prominent construction 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 (see 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.

Another proposal for building hash functions is the sponge construction [4], 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 processing the resulting state value using the underlying permutation.

Preneel et al. [5] 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, twelve schemes were deemed secure.

An application of hash functions is to compute MAC, which are used to authenticate data in cryptography. They are 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)* [6]. Computing a HMAC is defined as:

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

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 size as the state size of $\mathcal{H}$.

B. 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. 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 [7]. An attacker can 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)*. This attack computes hypotheses of the power consumption for each input and key candidate and compares them to the recorded power consumption of the device. Such a hypothesis can be computed by calculating the *Hamming weight (HW)* of a processed value, which is defined as the number of bits in a binary sequence set to $(1)_2$. 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 [7].

Profiling based attacks, such as template attacks, are another kind of power attacks. 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 recording the power consumption of the actual target and comparing it to the power consumption models, built in the profiling phase.

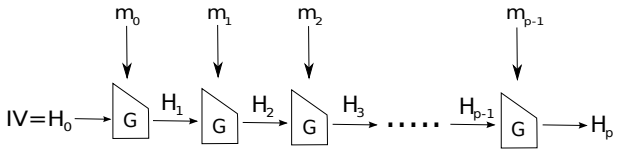

Fig. 1: Merkle-Damgård Construction

A method for performing a profiling based attack is the machine learning algorithm *Support Vector Machines (SVM)* [8]. 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. Hospodar et al. [9] 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 based approach. For the defined use cases, SVM and the template based approach performed similar in terms of accuracy, and thus Hospodar et al. deemed revealing cryptographic keys from power traces using SVM feasible.

In some cases it is more convenient to profile the Hamming weight of a processed value instead of the actual value. Using this approach, the number of power consumption models and thus the profiling complexity decreases[1]. However, since now multiple values are predicted, other methods are needed to reveal the actual value. Thus, profiling based power attacks were combined with *algebraic side channel attacks* [10], which build a system of equations from the algorithm and result in the actual key when solved.

III. Side Channel Attacks on Hash Functions used as HMAC

*1) **Side Channel Attacks Against HMACs Based on PGV Schemes**:* Okeya [11] 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 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, was deemed side channel resistant. In response to the SHA-3 competition, Okeya et al. [12] also presented a side channel analysis of several MAC functions.

*2) **Side Channel Analysis of Six SHA-3 Candidates**:* Benoît et al. [1] presented a side channel analysis of six round two SHA-3 candidates, used with a HMAC, and outlined vulnerabilities to which the DPA could be applied. Among the six analyzed candidates were the two finalists BLAKE and Grøstl [13][14]. For BLAKE-HMAC it was shown that the DPA could recover the keyed state, by exploiting the leakage of the modular addition and the XOR. For Grøstl-HMAC, the leakage of the S-box, which is the same as for AES, was

[1]Assuming that the Hamming weight model correctly describes the power consumption of the device.

targeted. Because the AES S-box is an operation, which is often targeted by side channel attacks, the keyed state was also deemed recoverable for Grøstl-HMAC.

IV. Analysis of the SHA-3 Candidates

We conducted a leakage analysis of the SHA-3 candidates and were able to identify major side channel vulnerabilities in each of the reference implementations. Following, we give an overview of our analysis for four out of five finalists of the SHA-3 competition and then sketch the respective attacks. The remaining candidate, BLAKE, has already been analyzed by Benoît et al. [1] 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. Additionally, an external frequency generator was used as clock for the device in order to reduce the offset in the time dimension.

The hypotheses and the measured power traces were compared using the Pearson correlation. In order 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 the correlation converged for a hypothesis. Each of the described DPAs was performed on 200 power traces.

In order to explain the attacks, we will state the hypothesis functions, used in the conducted DPAs. Note that the hypothesis functions are tailored to the target device and are not universally applicable. However, the generalized attack principle can be fit to other devices.

A. Skein

1) ***Description****:* Schneier et al. [15] proposed the hash function Skein, which is built by applying a novel construction, called *Unique Block Iteration (UBI)*, to the block cipher *Threefish*. UBI is similar to the Matyas-Meyer-Oseas construction, but additionally passes a tweak value to Threefish [16]. 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:

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

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 (see 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.

2) ***Side channel analysis****:* Since the data dependent leakage 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. Because Skein uses the Matyas-Meyer-Oseas construction, which is side channel secure (see [11]), we attacked Threefish by performing the DPA on the result of the modular addition between the keyed state and the input.

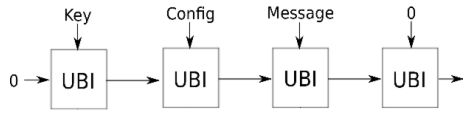

Fig. 2: Skein-MAC

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, we split the 64 bits of the keyed state value into eight blocks of eight bit each and attacked them independently. The hypothesis function h for the DPA is:

$$h(m_j)_c = HW(m_j + c), \text{ for } 0 \leq j < 8 \tag{3}$$

where $m_0 m_1 ... m_7 = M_i$ is the 64 bit block input message block and $c \in \{0, 1, ..., 255\}$ denotes the key candidate, for which the hypothesis is computed. 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 when attacking multiple blocks rises only linearly.

B. JH

1) ***Description****:* Hongjun Wu proposed the hash function JH [17], designed by using a novel construction method for building a collision resistant compression function from a block cipher. Additionally, he generalized the AES design methology, resulting in a block cipher E as base for the construction of JH. The JH construction processes the 512 bit message M_i and the 1024 bit state value H_i as follows:

$$H_{i+1} = E(H_i \oplus (M \mid\mid \{0\}^{512})) \oplus (\{0\}^{512} \mid\mid M). \tag{4}$$

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 subsequent linear transformation and permutation, which JH performs, are not detailed further

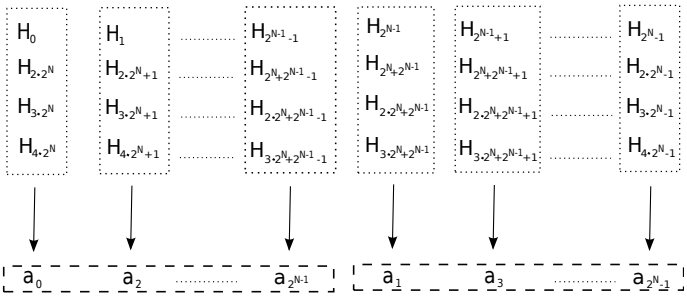

Fig. 3: JH Grouping

since they are not relevant for the side channel analysis. The suggested MAC function for JH is HMAC.

*2) **Side channel analysis**:* The following attack is applicable for the recovery of the inner and outer keyed state of JH-HMAC and will thus be described 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 attacked 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:

$$h_0(M)_c = HW(M \oplus c), \text{ for } c \in \{0,1\}^{512}. \quad (5)$$

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

After 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 input into 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 aim at recovering the two trailing bits $a_{i,2}$ and $a_{i,3}$. 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. The hypothesis function h_1 for the second DPA is:

$$h_1(a_{i,0}, a_{i,1})_c = HW(S_{C^0(i)}(a_{i,0} \mid\mid a_{i,1} \mid\mid c)), \quad (6)$$

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

C. Keccak

*1) **Description**:* Daemen et. al [18] proposed a family of permutation functions called Keccak, which are used by the sponge construction in order to build the corresponding hash function. The Keccak permutation familiy 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 performing:

$$A_{t,u,v} = H_{2^\lambda \cdot (5u+t)+v}, \text{ for } 0 \leq t,u < 5 \text{ and } 0 \leq v < 2^\lambda, \quad (7)$$

(see Figure 4). 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 it reveals the most information in terms of power attacks. The permutation θ is defined as:

$$\theta : A'_{t,u,v} = A_{t,u,v} \oplus \bigoplus_{i=0}^{4} A_{t-1,i,v} \oplus \bigoplus_{i=0}^{4} A_{t+1,i,v-1}. \quad (8)$$

Figure 5 visualizes the permutation θ. Roughly speaking, during the permutation, all elements of each column are XORed together and then XORed again with each matrix element. Keccak-MAC is computed by hashing $(K||M)$.

*2) **Side channel analysis**:* Performing a DPA on Keccak-MAC is complicated, 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 principle and give further possible courses of action.

The proposed attack is divided into two steps. At first it aims at recovering the bitrate of the sponge function by exploiting the XOR with the input. Secondly it targets the XOR operation, 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 also 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 and repeatingly attacking the next message bit until there is no correlation in a directly subsequent point in time. The reason for this procedure is that if two consecutive message bits are processed in the same permutation iteration, they will be processed by the XOR of the sponge construction successively, which can be verified by analyzing the correlation, which results 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 we suggest:

$$h_0(M_i)_c = HW(M_i \oplus c), \quad (9)$$

where $M_i \in \{0,1\}$ are the message bits, 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.

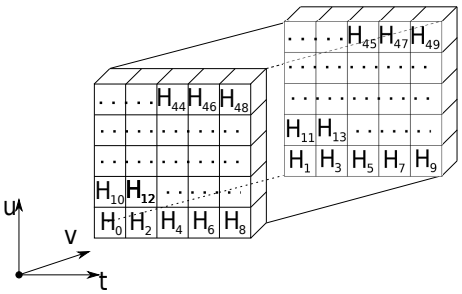

Fig. 4: Keccak assignment

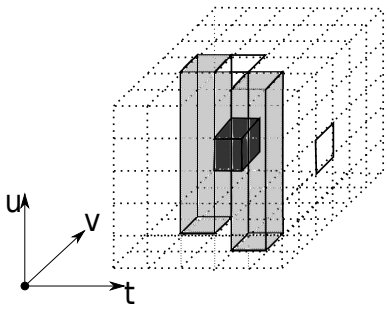

Fig. 5: Visualization of permutation θ for Keccak $f[50]$

The second step aims at recovering the remaining state value, i.e. the part of the bitrate, which is processed with the key, and the capacity of the sponge function. When the state value is input 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. 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 for this DPA is:

$$h_1(A_{i,j})_c = HW(A_{i,j} \oplus c), \tag{10}$$

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 all elements of the state matrix, and therefore the keyed state value, are known.

This attack becomes difficult when the number of recovered bits of the bitrate is smaller than $\frac{b}{5}$, i.e. 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 knew 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 small keys.

D. Grøstl

1) ***Description****:* Grøstl, proposed by Knudsen et al., is a Merkle-Damgård hash function, based on the block cipher standard AES [19], [14]. 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:

$$f(H_i, M_i) = H_{i+1} = P(H_i \oplus M_i) \oplus Q(M_i) \oplus H_i, \tag{11}$$

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. The operations were modified in order to cope with the increased state size. The difference between P and Q are the round constants in AddRoundConstant and the number of shifts in ShiftBytes.

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, because of the increased state size:

$$B_i = circ_i(02, 02, 03, 04, 05, 03, 05, 07), \text{ for } 0 \leq i < 8, \tag{12}$$

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

2) ***Side channel analysis****:* Benoît et al. [1] proposed a DPA against Grøstl-HMAC, which targeted the inner and outer keyed state. However, because of the vulnerability of AES to algebraic side channel attacks, we saw the necessity for a more thorough leakage analysis.

Until now the only considered side channel attack method for attacking SHA-3 candidates was the DPA. However, using the DPA, we can only recover a secret part of the input if we have multiple measurements and prior knowledge of some parts of a varying input. This strongly limits the applicability of a DPA when attacking hash functions. Therefore, we were interested in the recovery of the input to a hash function call without requiring any prior knowledge about the input.

For this scenario, a profiling based attack seemed the right choice. Using a profiling based attack, we can recover information about the intermediate values. In order to exploit these recovered information, we set up a system of equations for P and Q, which set the intermediate values 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 values, and lastly insert the recovered HW into the system of equations.

For the task of determining the HW of the intermediate values, we chose SVM because of its good results in [9]. Since there has not been an approach of classifying 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 best correlated points of a power trace as well as their sum, together with an RBF kernel and a σ of 1. This setup yielded an accuracy of 97.1% 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. SubBytes and MixBytes were chosen because they set the most constraints on the processed values. Simulations indicated that if 200 HW of the total 320 HW of the equation system were known, the processed data could be recovered.

The attack was performed on the ATMega-256-1 using the reference implementation of Grøstl-256 with a 8×8 state

matrix. In total we 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, ten traces were recorded, always processing the same message, and their mean was input into the constructed SVMs in order to classify the HW of the profiled operations. These HW were inserted into the algebraic system, which was solved by verifying the conditions, imposed by the equations.

V. 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 the 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 the performed side channel attacks against the reference implementations of the SHA-3 candidates do not tell how hard it is in practice to successfully attack SHA-3 candidate implementations, since the reference implementation is designed with the focus on 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 outline the necessity of side channel resistant hash functions and to identify the operations, which an attack would exploit, so that they can be secured using countermeasures.

In terms of countermeasures, some research has already been conducted. In [20] Hoerder et al. performed an evaluation of hash functions, including Skein, BLAKE, and Keccak, on a power analysis resilient processor architecture. For Keccak there have been some notes on side channel countermeasures [21] as well as proposals for masked Keccak implementations [22][23]. In case of Grøstl, many side channel countermeasures, developed for the AES, can be adopted.

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 so that all application scenarios are covered. 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. 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] O. Benoît and T. Peyrin, "Side-channel analysis of six SHA-3 candidates," in *Proceedings of the 12th international conference on Cryptographic hardware and embedded systems*, ser. CHES'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 140–157.

[2] TU Graz, "The SHA-3 Zoo." [Online]. Available: http://ehash.iaik.tugraz.at/wiki/The_SHA-3_Zoo

[3] J. Kelsey, "How to Choose SHA-3." [Online]. Available: http://www.lorentzcenter.nl/lc/web/2008/309/presentations/Kelsey.pdf

[4] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche, "Cryptographic sponge functions," Submission to NIST (Round 3), 2011. [Online]. Available: http://sponge.noekeon.org/CSF-0.1.pdf

[5] B. Preneel, R. Govaerts, and J. Vandewalle, "Hash functions based on block ciphers: a synthetic approach," in *Proceedings of the 13th annual international cryptology conference on Advances in cryptology*. New York, NY, USA: Springer-Verlag New York, Inc., 1994, pp. 368–378.

[6] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication," United States, 1997.

[7] S. Mangard, E. Oswald, and T. Popp, *Power analysis attacks - revealing the secrets of smart cards*. Springer, 2007.

[8] C. J. C. Burges, "A Tutorial on Support Vector Machines for Pattern Recognition," *Data Min. Knowl. Discov.*, vol. 2, no. 2, pp. 121–167, 1998.

[9] G. Hospodar, E. D. Mulder, B. Gierlichs, I. Verbauwhede, and J. Vandewallei, "Least squares support vector machines for side-channel analysis," in *2nd Workshop on Constructive Side-Channel Analysis and Secure Design, COSADE*, 2011.

[10] M. Renauld, F.-X. Standaert, and N. Veyrat-Charvillon, "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*, ser. CHES '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 97–111.

[11] K. Okeya, "Side Channel Attacks Against HMACs Based on Block-Cipher Based Hash Functions," in *Information Security and Privacy*, ser. Lecture Notes in Computer Science, L. Batten and R. Safavi-Naini, Eds. Springer Berlin / Heidelberg, 2006, vol. 4058, pp. 432–443.

[12] P. Gauravaram and K. Okeya, "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*, ser. ICICS, 2008, pp. 111–127.

[13] J.-P. Aumasson, L. Henzen, W. Meier, and R. C.-W. Phan, "SHA-3 proposal BLAKE," Submission to NIST (Round 3), 2010.

[14] P. Gauravaram, L. R. Knudsen, K. Matusiewicz, F. Mendel, C. Rechberger, M. Schläffer, and S. ren S. Thomsen, "Grøstl – a SHA-3 candidate," Submission to NIST (Round 3), 2011.

[15] N. Ferguson, S. Lucks, B. Schneier, D. Whiting, M. Bellare, T. Kohno, J. Callas, and J. Walker, "The Skein Hash Function Family," Submission to NIST (Round 3), 2010.

[16] M. Liskov, R. L. Rivest, and D. Wagner, "Tweakable block ciphers," in *Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology*, ser. CRYPTO '02. London, UK, UK: Springer-Verlag, 2002, pp. 31–46.

[17] H. Wu, "The Hash Function JH," Submission to NIST (round 3), 2011.

[18] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche, "The Keccak reference," Submission to NIST (Round 3), 2011.

[19] Announcing The Federal, "Processing Standards Publication 197."

[20] S. Hoerder, M. Wojcik, S. Tillich, and D. Page, "An evaluation of hash functions on a power analysis resistant processor architecture," in *Workshop in Information Security Theory and Practice - WISTP 2011*. Springer-Verlag LNCS 6633, June 2011, pp. 160–174.

[21] The Keccak Team, "Note on side-channel attacks and their countermeasures." [Online]. Available: http://keccak.noekeon.org/NoteSideChannelAttacks.pdf

[22] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche, "Building power analysis resistant implementations of Keccak," in *Second SHA-3 Candidate Conference*, August 23-24, 2010.

[23] E. Alemneh, "Sharing nonlinear gates in the presence of glitches," August 2010, Master's Thesis. [Online]. Available: http://essay.utwente.nl/59599/