Side-channel Analysis of Grøstl and Skein

Christina Boura, Sylvain Lévêque, David Vigilant

Side-channel Analysis of Grøstl and Skein

Christina Boura* †, Sylvain Lévêque*, David Vigilant*
* *Gemalto*
6 rue de la Verrerie, 92190 Meudon, France
{sylvain.leveque, david.vigilant}@gemalto.com
† *SECRET Project-Team - INRIA Paris-Rocquencourt*
Domaine de Voluceau - B.P. 105 - 78153 Le Chesnay Cedex - France
christina.boura@inria.fr

***Abstract*—This work[1] provides a detailed study of two finalists of the SHA-3 competition from the side-channel analysis point of view. For both functions when used as a MAC, this paper presents detected strategies for performing a power analysis. Besides the classical HMAC mode, two additionally proposed constructions, the envelope MAC for Grøstl and the Skein-MAC for Skein, are analyzed. Consequently, examples of software countermeasures thwarting first-order DPA or CPA are given. For the validation of our choices, we implemented HMAC-Grøstl, HMAC-Skein as well as countermeasures on a 32-bit ARM-based smart card. We also mounted power analysis attacks in practice on both unprotected and protected implementations. Finally, the performance difference between both versions is discussed.**

***Keywords*-side-channel, HMAC, SHA-3, countermeasures**

I. INTRODUCTION

Hash functions are often called the "swiss army knives" of cryptography. Their use in password protection, in data integrity checks or in digital signatures demonstrates the necessity of the existence of hash functions with good security properties. In 2004, Wang *et al.* [1] presented a number of devastating collision attacks for many widely used functions, such as MD5 and SHA-1. In response to these attacks, NIST launched in 2007 a public competition aiming at defining a new hash function standard. This competition, called the SHA-3 contest, should come to an end with the announcement of the winner in the second semester of 2012. Currently, only five candidates remain: BLAKE, Grøstl, JH, KECCAK and Skein.

One of the most important applications of a hash algorithm is the message integrity and authentication, *i.e.* the recipient of a message can verify that the received message is identical to the one sent and at the same time can authenticate its author. In this case, the two parties agree on a secret key K, and this key is then used in the hash computation together with the message, to produce the *message authentication code* (MAC). Many hash-based MAC constructions have been proposed [2], [3]. The HMAC one [3] is probably the most popular among them. Consequently, their use in MAC constructions makes them a target for side-channel attacks [4].

Regarding the SHA-3 competition, NIST required that all the submitted functions possess a secure HMAC or other MAC mode. In parallel, it was specified that side-channel issues would be taken into consideration for the final decision. For all these reasons, analyzing the resistance against side-channel attacks [5], [6] of the remaining candidates has become an important matter [7]. In this direction, Benoît and Peyrin presented in [8] an analysis of the resistance against side-channel attacks in a MAC setting of six second round candidates. In their work, a theoretical analysis exhibits the best selection functions for each candidate. Then, these functions were implemented on an FPGA in order to measure the electromagnetic leakage. In a more recent work [9], the resistance of four out of five third round SHA-3 candidates against side-channel attacks was analyzed, and target operations were proposed as well.

This paper presents an analysis of the side-channel resistance of two SHA-3 finalists, Grøstl [10] and

[1] This work is partially supported by the French Agence Nationale de la Recherche through the SAPHIR2 project under Contract ANR-08-VERS-014.

DOI 10.1109/SPW.2012.13

Skein [11]. Both functions have been implemented on a smart card and their HMAC modes have been attacked by CPA. In parallel, a first serious analysis for the possible countermeasures on both functions against first-order DPA and CPA attacks is presented. After recalling two basic MAC modes in Section II and reminding the basic principles of a correlation power analysis in Section III, the main results on Grøstl and Skein are presented in Sections IV and V.

The attack setting: Grøstl-256, Skein-512-256 and their respective HMAC were implemented on a 32-bit ARM architecture smart card, running at 8MHz. The security settings of this smart card include the activation of all hardware sensors and of a random current generator. Its CPU is known to leak information over power with the Hamming weight model, at a relatively low level regarding industry standards. The aim of this paper is not to reach optimal absolute timings for the execution of the two SHA-3 candidates. Therefore, in both cases, the reference implementation proposed by the designers was employed. The purpose of this work is not to minimize the number of curves needed for every attack. Hence, we chose to set the number of recorded waveforms to 5000 for both algorithms, for both non-secure and secure implementations. This paper focuses on providing a comparison between the plain and secure versions using the same reference model, and evaluating the extra cost to reach a secure implementation against first-order statistical power analysis. First-order Correlation Power Analysis is used to exploit the power leakage and reveal the targeted secret values. Examples of cost-effective countermeasures thwarting this threat are proposed and validated. A more complete security analysis, for example against second-order DPA or fault attacks, would need more investigation and overcomes the scope of the present work.

II. HMAC AND ENVELOPE MAC

A Message Authentication Code (MAC) based on a hash function is frequently used to check the authenticity and the integrity of a message sent over an insecure channel.

One of the most popular MAC constructions is HMAC, presented by Bellare *et al.* in [3].

A HMAC based on the hash function H is defined as follows:

$$\text{HMAC-H}(K, M) =$$
$$H((\overline{K} \oplus opad)||H((\overline{K} \oplus ipad)||M)).$$

Here, *ipad* and *opad* are two constants the size of a message block, while $\overline{K}$ is the key K padded with 0's until reaching the block size. Longer keys are first hashed with H.

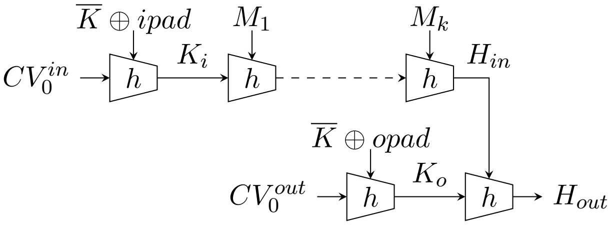


Figure 1: The HMAC construction.

It is easy to see from Figure 1, that the first block for each call to h is a constant value that depends only on K. In some implementations though, to gain in performance, the values

$$K_i = h(\overline{K} \oplus ipad) \text{ and } K_o = h(\overline{K} \oplus opad)$$

are precomputed and stored on the device.

Our attack will attempt to recover the values of K_o and K_i. Knowing them allows to forge the MAC whatever the message M and the value of H_{in} are. The techniques to recover K_o and K_i are similar, for this reason this paper will only focus on attacking K_i.

A MAC construction, that had been originally proposed by Tsudik [2] much earlier than the HMAC scheme and repaired later by Yasuda [12] after an attack on the original scheme, is the so-called *envelope* MAC. It was designed to combine both the *secret prefix* construction, *i.e.* $\text{MAC}_K(M) = H(K||M)$ and the *secret suffix* construction, *i.e.* $\text{MAC}_K(M) = H(M||K)$. The repaired version of *envelope* MAC is simply

$$\text{MAC}_K(M) = H(\overline{K}||\overline{M}||K),$$

where $\overline{K}$ and $\overline{M}$ are the padded secret key K and the padded message M respectively. In this way, the key and the message blocks are treated separately.

Envelope MAC has been proposed as the dedicated MAC construction for Grøstl [10].

In the HMAC construction, obtaining K_i and K_o was enough to forge the MAC. But for a successful

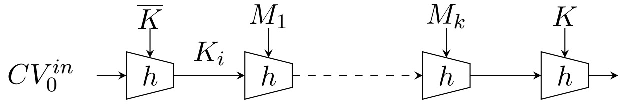


Figure 2: The envelope MAC construction

attack of the envelope MAC, one has to imperatively recover the secret key K. However, in the DPA-CPA scenario, the amount of effort for an efficient attack is equal to the effort needed for forging a HMAC, as again the attack must be set up in two steps. First, the value of K_i must be recovered, then the insertion of the key to the last compression function must be targeted by processing many different messages M.

III. Correlation Power Analysis

Side-channel attacks are a class of physical attacks against cryptographic implementations, where one tries to exploit the information leaked from a device executing a cryptographic algorithm. Many different leakages can be measured, however the power consumption of a device is one of the most frequently employed. It can easily be observed with an oscilloscope and used to make deductions about the secret information that is manipulated.

An important class of power analysis attacks is the statistical power analysis. It was first introduced by Kocher *et al.* [5], in the form of *differential power analysis* (DPA) and later the *correlation power analysis* (CPA) improved the above technique [6]. They both rely on a leakage model, several of them were proposed. The *Hamming weight model*, introduced in [13], [5] is probably the most classical among them. It involves an affine relationship between the Hamming weight of the manipulated data and the power consumption:

$$Y = aHW(X) + b,$$

where X is the information being manipulated and HW stands for the Hamming weight.

The first step of a statistical power analysis attack on a hash-based MAC is to choose a target operation. This operation must be of the form $f(\alpha, k) = \beta$. k is a part of K_i or K_o in the case of an HMAC and, in the case of an envelope MAC when attacking the last compression function, a part of the key K. On the other hand, α is a known random value, such as part of the message or the chaining value being processed. The function f, frequently called *selection function*, can be any operation mixing secret and public data, such as an XOR operation, a modular addition $\boxplus$, or a substitution table.

To put a CPA-type attack in place, the adversary runs the target device N times, with N different messages and captures for each message a power consumption waveform. For each power curve, the attacker will try to predict the Hamming weight of the word being manipulated at a chosen point in time, by calculating the Pearson correlation coefficient. This will be done for every possible value of k, and a CPA trace will equally be generated. The correlation should be maximized for the correct key guess and thus a peak should appear at that moment of time.

IV. Grøstl

Grøstl [10] is a family of iterated hash functions based on a compression function f. The variant returning 256 bits is denoted by Grøstl-256. The compression function of Grøstl-256 is iterated as follows. First, the message m to hash is padded and cut into 512-bit blocks, $m_1, \ldots, m_t$. After this, given an initial value $iv = h_0$, every message block is processed sequentially:

$$h_i = f(h_{i-1}, m_i), \text{ for } 1 \leq i \leq t.$$

Finally, an output transformation Ω

$$\Omega(x) = trunc_n(P(x) \oplus x),$$

is applied to h_t, where $trunc_n(x)$ is the function discarding from x all but the n first bits.

The compression function f is built out of two large distinct permutations P and Q:

$$f(h, m) = P(h \oplus m) \oplus Q(m) \oplus h.$$

Grøstl iterated permutations P and Q are based on the Rijndael block cipher. They are applied to a 512-bit state represented by a square matrix of bytes A, with eight rows and columns. In every round R, a total of four transformations inspired from the AES are performed:

$$\begin{aligned} R \;=\; & \texttt{MixBytes} \circ \texttt{ShiftBytes} \circ \\ & \texttt{SubBytes} \circ \texttt{AddRoundConstant}. \end{aligned}$$

The `AddRoundConstant` transformation adds a round-dependent constant to every byte of the state matrix A.

The `SubBytes` transformation consists of a non-linear substitution applied to every byte separately. It is identical to the AES `SubBytes` transformation.

`ShiftBytes` cyclically shifts the bytes within a row to the left by a number of positions.

The `MixBytes` operation transforms every column of the state matrix independently in a linear way.

The number of rounds is set to ten for Grøstl-256.

A. Side-channel analysis of Grøstl

This section provides a power analysis of Grøstl when implemented as a MAC. HMAC is clearly the most employed hash-based MAC, but as the designers of Grøstl suggest the envelope MAC construction, both of these designs have been studied. The first step in this analysis consists in simply identifying which operations are sensitive against a statistical power attack and must therefore be protected. We realized such attacks on the non-secure HMAC-Grøstl for two main reasons. The first one was to practically confirm our theoretical assumptions by showing that some leakage of the secret information takes place in reality. The second one was to see at what point the secret key could be recovered, as the success of such an attack depends very often on the targeted architecture and selection function.

This analysis starts by examining HMAC-Grøstl. Our aim is to recover K_i and K_o, and the procedure to followed is similar for both. In order to retrieve the secret information, we have to identify the selection functions where secret and public data are mixed together.

Three such simple operations can be identified. The first one is the XOR between h_{i-1} and m_i. In the HMAC setting, h_{i-1} will contain K_i when the first message block is processed. The second one is the first `SubBytes` operation. A similar analysis was done for Grøstl in [8], where the authors stated that the `SubBytes` operation yields better correlation properties than the XOR operation. In our approach, both operations are targeted, in order to compare the real difference in their efficiency.

A third possible target is the XOR between h_{i-1}, $Q(m_i)$ and $P(h_{i-1} \oplus m_i)$ to compute the next chaining value h_i. However, the possibility to attack this operation depends on the way it is implemented. Three possible ways exist. The first one consists in computing the XOR between $Q(m_i)$ and $P(h_{i-1} \oplus m_i)$ and then XORing this result to h_{i-1}:

$$\begin{aligned} tmp &= Q(m_i) \oplus P(h_{i-1} \oplus m_i) \\ h_i &= h_{i-1} \oplus tmp. \end{aligned}$$

The second way is to compute the XOR of $P(h_{i-1} \oplus m_i)$ and h_{i-1} and XOR this to $Q(m_i)$:

$$\begin{aligned} tmp &= P(h_{i-1} \oplus m_i) \oplus h_{i-1} \\ h_i &= Q(m_i) \oplus tmp, \end{aligned}$$

and the third one consists in first combining h_{i-1} with $Q(m_i)$ and then XORing $P(h_{i-1} \oplus m_i)$ like this:

$$\begin{aligned} tmp &= Q(m_i) \oplus h_{i-1} \\ h_i &= P(h_{i-1} \oplus m_i) \oplus tmp. \end{aligned}$$

The first two ways of implementing this operation do not provide any apparent threat for the hash function, as $P(h_{i-1} \oplus m_i)$ is an unknown changing data. In the third implementation, the first operation is similar to the first XOR between m_i and h_{i-1}, using $Q(m_i)$ instead of m_i. It is therefore CPA-sensitive. The reference code provided by the designers does not use the third implementation and is therefore CPA-proof.

A special point is that both sensitive operations *i.e.* the initial XOR and the first Sbox layer, can be captured in the same power trace with a good signal resolution. From an attacker's point of view, it means that only one set of curves is needed. This can be seen in Figure 3, where both attack locations can be identified.

5000 curves are captured and then analyzed with respect to correlation power analysis. Clear leakages can be observed for both operations and all the key bytes could be successfully recovered. However, as expected, the correlation peaks for the correct key guess are clearer for the `SubBytes` operation than for the XOR. The Figure 4 shows the results of a successful CPA against the `SubBytes` operation. As explained in the introduction, the scope of this analysis is not to find the minimal number of curves needed in order to entirely recover the key. Our approach aims at demonstrating and highlighting operations in Grøstl that leak secret information in practice and thus showing that masking these operations is of major importance.

Regarding the envelope MAC, the preliminary analysis does not give much different results from the side-channel point of view. The only significant difference

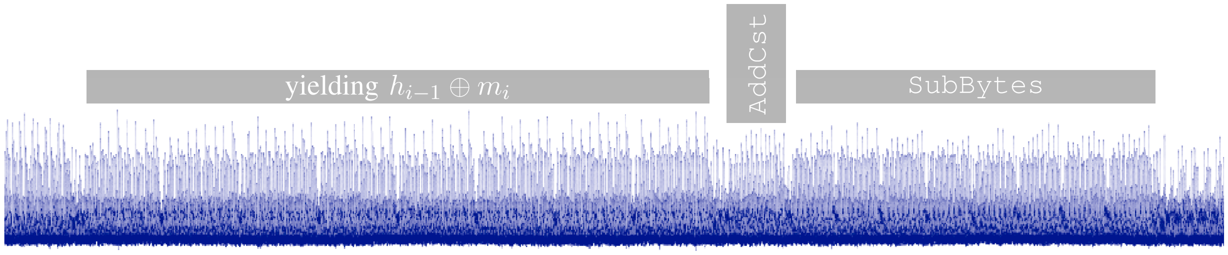


Figure 3: SPA of the two attack areas in HMAC-Grøstl: message entry and `SubBytes` operation

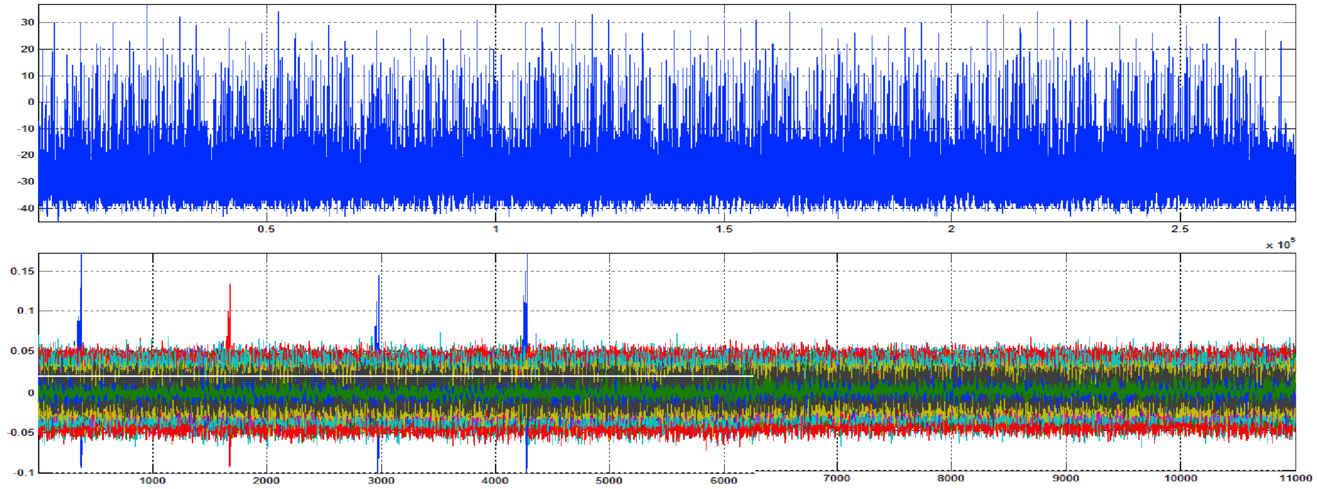


Figure 4: CPA on HMAC-Grøstl `SubBytes` operation (four first bytes of targeted value)

is that in the last call of the compression function, the roles of the secret and the public information are interchanged. As these roles are entirely symmetric, this does not change anything in the previous attack scenario, except that it permits to recover the secret key. If this key is also used for some other applications of the smart card, the attack can be more devastating than in the HMAC case, where the original key is not recoverable.

In the sequel, countermeasures for Grøstl are proposed, protecting the identified sensitive operations. These countermeasures apply for both HMAC and envelope MAC settings.

B. *Countermeasures for Grøstl*

We come up with very simple countermeasures that mask the sensitive data for all rounds of Grøstl. These countermeasures should apply against any first-order statistical power analysis. As mentioned before, the critical operations are the XOR between the incoming message block and the chaining value, and then the `SubBytes` operation.

In order to protect the XOR, a Boolean mask R of 512 bits is generated once. This mask is XORed to the chaining value, and the feed-forward naturally re-injects it at the beginning of every compression function. This mask R, called global, should be deleted by simply re-XORing it to the state just before the final truncation. Its propagation can be seen in Figure 5, together with the unprotected hash computation. On the other hand, in order to go through the Sbox layer, the global mask must be removed at the beginning of every permutation P. Thus, another type of protection is needed to mask the Sbox computation and the rest of the permutation. For this, many possible solutions exist, coming mostly from the study of the protection of AES against side-channels. An easy method for masking an Sbox S, that can be seen in practice as a table lookup operation, is to mask the table itself. For this, an input mask u and an output mask v can be used, and the masked Sbox S' is computed in terms of S, u and v as

$$S'(x \oplus u) = S(x) \oplus v.$$

In practice, the masks u and v are obtained at the beginning of the algorithm. The Sbox S' is then

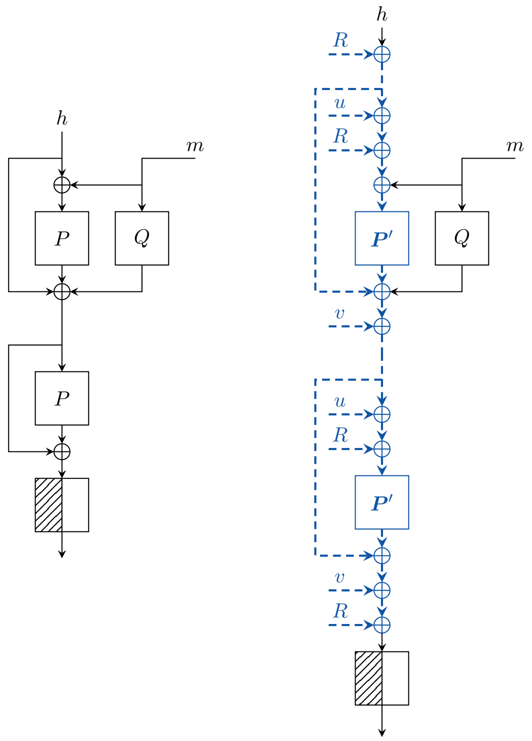


Figure 5: Side-by-side comparison of standard and secured Grøstl

constructed and stored in RAM, together with u and v, for the remaining computations.

The proposed countermeasures would not hold in the case of a higher-order DPA or CPA, where the sensitive value should be split in more than two random parts. Here one could have also used the duplication method [14] to protect this stage.

This new mask, called local, should be applied to the state before the global mask is deleted. In this way, no sensitive value appears unmasked at any moment of the computation. In Figure 5, P' symbolizes the permutation P in which the above countermeasures are implemented.

One can verify that the previously observed leakage disappears when trying to attack the secured implementation, as shown in Figure 6.

The next section presents an analysis of Skein. To have a relevant comparison between the two SHA-3 candidates, appropriate settings are chosen for the internal state size of Skein.

V. SKEIN

Skein [11] is a hash function family based on the tweakable block cipher Threefish. Three different internal state sizes are available: 256, 512 and 1024 bits. The key size of the block cipher is equal to the block size and the tweak value is 128 bits for all three versions. The tweak value for each block encodes the number of bytes processed so far, together with some other information.

This work analyzes only Skein-512-256, *i.e.* the version having 512 bits of internal state and outputting 256 bits of digest. Only this specific version is therefore described.

Threefish uses only three mathematical operations: XOR, modular addition and rotations by a constant on 64-bit words. Its basic function, called MIX, is described in Figure 7a.

Skein-512-256 is composed of 72 rounds of basic operations. Every round consists of four parallel applications of the MIX function, followed by a permutation of the eight words of the state. A subkey is injected every four rounds. One round is shown in Figure 7b.

All the operations are done on 64-bit words and both the internal state S and the key are composed of eight words. Let t_0 and t_1 define the two words of the tweak value and let $k_0, \ldots, k_7$ be the eight key words. The first subkey is given by

$$s^0 = (k_0, k_1, k_2, k_3, k_4, k_5 + t_0, k_6 + t_1, k_7),$$

where the additions are made modulo 2^{64}. The compression function of Skein is defined as

$$h_i = E_{h_{i-1},T_i}(m_i) \oplus m_i,$$

where $E_{K,T}(P)$ is the Threefish cipher, h_{i-1} is the previous chaining value, T_i the tweak and m_i the message block.

The chaining mode used is the Unique Block Iteration (UBI). For standard hashing, a configuration block is processed through the compression function before hashing the message and another call to the compression function is made after all the message blocks have been handled. A unique 128-bit tweak value is used for every compression function call.

A. Skein-MAC

The submission document [11] mentions that Skein can naturally be used in HMAC mode, but this application is not suggested by the authors because of the

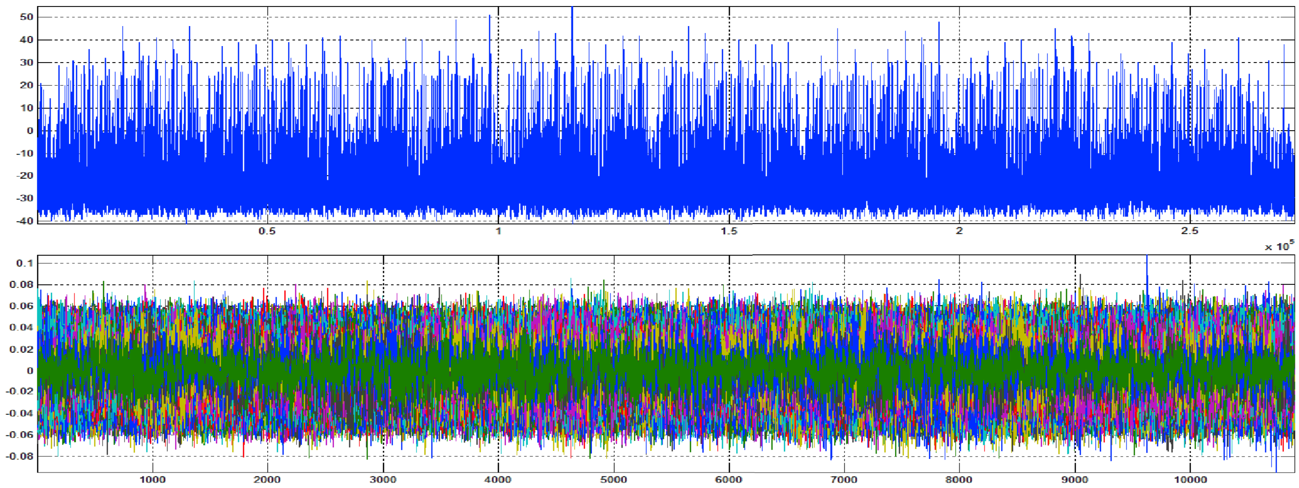
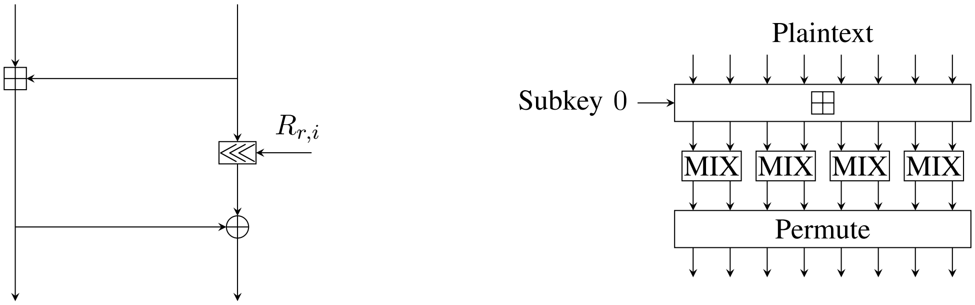


Figure 6: CPA on secure HMAC-Grøstl `SubBytes` operation (first four bytes of targeted value)



(a) The MIX operation

(b) One round of Threefish-512

Figure 7: Elements of the Skein function

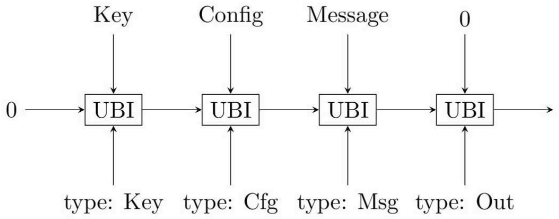


Figure 8: The Skein-MAC construction

inefficiency for short messages. Alternatively, a simple method for turning Skein into a MAC can be seen in Figure 8.

It consists in first processing the key with 0 as a chaining value, and then the configuration block.

B. Side-channel analysis of Skein

An analysis of Skein-512-256 when implemented as a MAC is now provided. We have looked for attacking strategies, for both HMAC-Skein and Skein-MAC. Exactly the same sensitive operations in both MAC designs can be identified. For the HMAC, the attack would consist in recovering first K_i and then K_o. In Skein-MAC, only one secret value, the UBI output of the configuration block treatment, must be recovered to completely forge the MAC. Remarks made for HMAC-Skein should therefore apply to Skein-MAC as well.

As also mentioned in [9], the easiest operation to attack in terms of statistical power analysis is the modular addition between the message and the first subkey. In the HMAC setting, if $m_0, \ldots, m_7$, are the eight 64-bit words of the first message block and $K_0, \ldots, K_7$ denote the eight words of equal length of

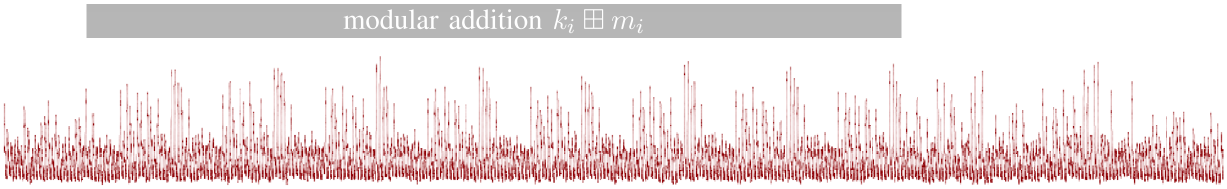


Figure 9: SPA of the attack area in HMAC-Skein: message entry

the first subkey, this operation is simply

$$K_i \boxplus m_i, \text{ for } i = 0, \ldots, 7.$$

Consequently a CPA attack on the HMAC-Skein is mounted, targeting these eight additions and trying to completely recover the secret value word by word. As previously, 5000 curves are analyzed. As one can see in Figure 10, the results of this experimentation are finally much more surprising than expected. Indeed, if we denote by $(b_0, \ldots, b_7)$ the bytes of a 64-bit word of the first subkey, only the bytes b_0, b_4, b_5, b_6 and b_7 could be recovered in practice. But even out of these five bytes, the highest peak gives the right value only for b_0, while for the others, extra effort is required to determine the correct value, among a few other wrong peaks.

The reason for this unexpected behavior might come from the way a modular addition of two 64-bit values is treated on a 32-bit architecture. This problem will probably be the subject of a future work. However, even if the key is not entirely recovered, the recorded leakage is clearly significant enough to make the protection of this addition necessary.

C. *Countermeasures*

The modular addition between the message and the first subkey must be protected, but the diffusion provided by the MIX operation through a single round is not significant enough. Masking ARX-based functions (*i.e.* functions based on Additions, Rotations and XORs) requires conversions between Boolean and arithmetic values, which can be very expensive. Therefore a trade-off between security and performance should be found. We decided that protecting the first four rounds would ensure a good diffusion of the secret, withdrawing all the possible first-order DPA or CPA on further rounds, with a limited performance impact.

A method for switching between Boolean and arithmetic masking should be chosen. An efficient algorithm for converting from Boolean to arithmetic masking was proposed by Goubin in [15]. A method for the inverse operation, *i.e.* converting from arithmetic to Boolean masking was also proposed in his paper. The number of operations for this conversion depends on the size of the data to convert, which is in our case 64-bit words. Another algorithm, for this same conversion, was proposed in [16]. In this approach, to avoid the high number of expensive operations, a large masking table is implemented, thus speed is gained at the expense of memory.

Since Skein is a quite fast candidate, keeping memory resources low while requiring more processing time is a more interesting trade-off. For the above reasons, Goubin's solution [15] has been implemented for four rounds of Skein.

As the permutation used for Skein is such that the odd and the even indexes are not mixed together, the same arithmetic mask R_o for the words with an odd index, and the same arithmetic mask R_e for the even ones, can be used. It interestingly reduces the number of calls to the random number generator, as well as the amount of RAM required for storing the mask values.

Furthermore, as already mentioned, the most expensive conversion is the arithmetic to Boolean one. In our implementation the performance ratio between the two conversions is roughly 16. In the straightforward approach, as the first key injection is a modular addition, it feels natural to use two arithmetic masks. This implementation, depicted in Figure 11, requires two arithmetic to Boolean conversions for every MIX computation, *i.e.* 16 for each round. In order to avoid this high number of arithmetic to Boolean conversions, we propose to apply the following tweak just before entering the first layer of MIX operations.

Let R_o be the arithmetic mask protecting the subkey insertion of the words with odd index. Before the MIX operation, an arithmetic to Boolean conversion is performed for the odd branch of all the MIX operations. In

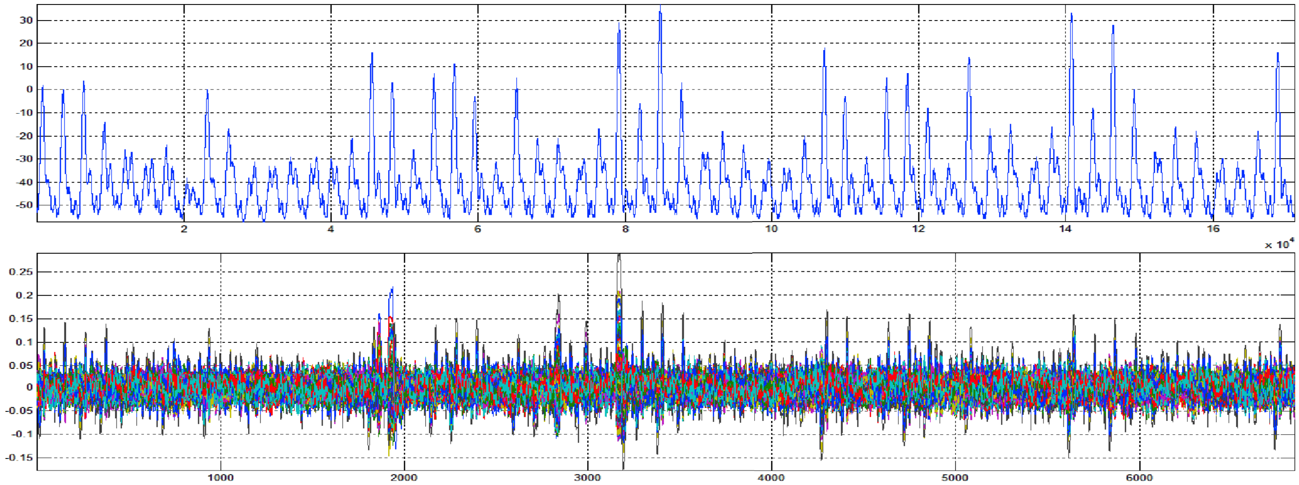


Figure 10: CPA on the modular addition of Skein

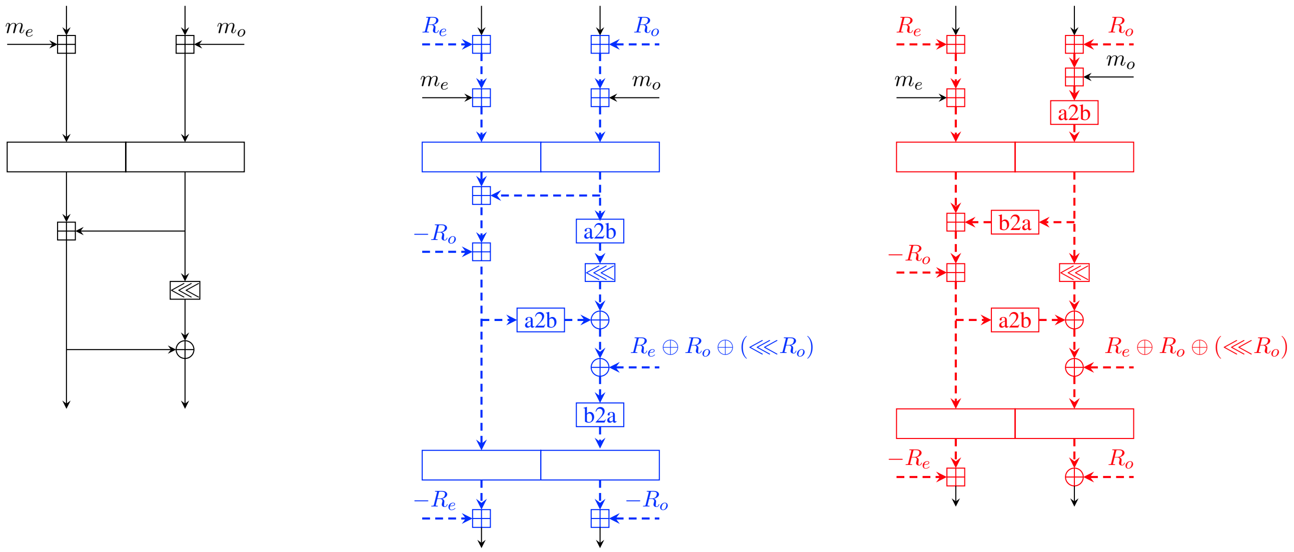


Figure 11: Comparison of standard and secured Skein (without and with the tweak)

this way, the left branch of every MIX operation will be protected by an arithmetic mask, while the right part will be protected by a Boolean one. Then, the following operations will be performed for all the 32 MIX operations of the first four rounds.

A Boolean to arithmetic conversion will be applied to the right branch of the MIX before the modular addition and a Boolean to arithmetic one will be done to the left branch before the XOR, as this can be seen in Figure 11. With this method, only one arithmetic to Boolean conversion is performed inside every MIX. This means that with this optimization, 8 arithmetic to Boolean conversions are needed before applying the first round transformation, and 32 are needed inside the first four rounds, rather than 64 that were needed before. A performance gain of around 30% on the total HMAC computation can be observed.

The CPA analysis mounted on the protected version only shows a high peak that corresponds to the null subkey, *i.e.* the correlation with the message.

VI. Performance Analysis

One of the scopes of this paper is to provide a comparison between two of the finalists of the SHA-

Algorithm	Timings at 8MHz		Extra RAM		Extra code
	reference code	secured code	static	stack	
HMAC-Grøstl	453 ms	486 ms (+7.2%)	+325 bytes	0	+688 bytes
HMAC-Skein	77.7 ms	155 ms (+100%)	0	+32 bytes	+3484 bytes

Table I: Secured code overhead for timing, RAM consumption and code size when hashing one block of message

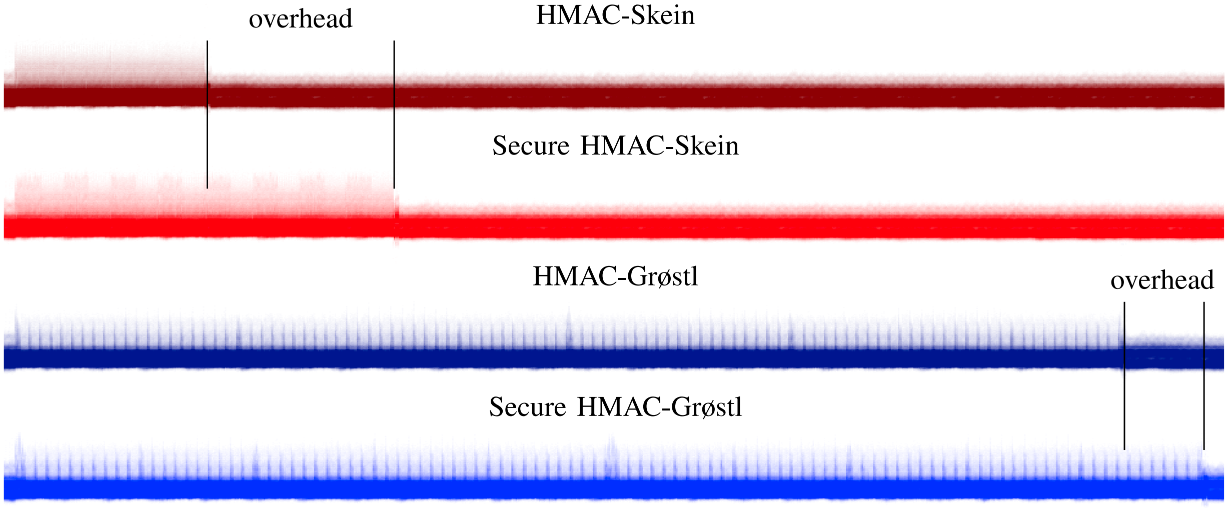


Figure 12: Power consumption for reference and secured implementations of HMAC-Skein and HMAC-Grøstl

3 competition. Even if the proposed MAC mode for both algorithms was not the HMAC construction, we chose to implement this mode for Skein and Grøstl for reasons of analogy, as the HMAC is until nowadays the most usually employed MAC. In this way, a comparison between the two candidates can be kept.

The difference in the time required for the computation of each of the HMACs before and after the application of the countermeasures, on one block of message, can be an example of such a comparison. These measurements can be seen in Table I. In this table, the amount of RAM needed to implement the security parameters is equally mentioned, as well as the extra code size required. For a better visualization of the results, the same timing results are presented in Figure 12. Here, the power consumption curves produced by an oscilloscope during the computations of HMAC-Grøstl and HMAC-Skein for 1-block messages, for both protected and unprotected implementations can be seen. One can in particularly observe that our performance ratio between HMAC-Skein and the HMAC-Grøstl roughly came to 3 from 6 after protection.

VII. Conclusion

The winner function of the SHA-3 competition will be announced soon. This function will be without doubt implemented on various smart cards and other devices. A discussion about the physical resistance of the candidates and the possible countermeasures has started. Here, two of the finalists of this competition were examined, and more material to this analysis was brought.

More precisely, this paper analyzed the resistance of Grøstl and Skein against first-order CPA. For both of them when used as a MAC, all the possible target operations have been highlighted. In order to validate the sensitivity of operations, HMAC-Grøstl and HMAC-Skein were implemented on a 32-bit ARM-based chip, 5000 power curves were collected for every operation, and a CPA was mounted on each of them. In the case of Grøstl, the correct key can be recovered for both selection functions, *i.e.* the XOR and the Sbox layer, even if the first one is a linear operation. On the contrary, it was not possible to recover the entire key for Skein.

Then countermeasures were proposed for both algorithms. For Grøstl, the proposed masking operations are quite efficient and entirely protect HMAC-Grøstl (and in consequence Grøstl envelope MAC) for the cost of an extra 7% of its initial speed. For Skein, the situation is much more complicated, as it is necessary to switch many times between arithmetic and Boolean masking, which is very expensive.

References

[1] X. Wang and H. Yu, "How to Break MD5 and Other Hash Functions," in *EUROCRYPT 2005*, ser. Lecture Notes in Computer Science, vol. 3494. Springer, 2005, pp. 19–35.

[2] G. Tsudik, "Message Authentication with One-Way Hash Functions," in *INFOCOM 1992*, 1992, pp. 2055–2059.

[3] M. Bellare, R. Canetti, and H. Krawczyk, "Keying Hash Functions for Message Authentication," in *CRYPTO 1996*, ser. Lecture Notes in Computer Science, vol. 1109. Springer, 1996, pp. 1–15.

[4] K. Okeya, "Side Channel Attacks Against HMACs Based on Block-Cipher Based Hash Functions," in *ACISP 2006*, ser. Lecture Notes in Computer Science, vol. 4058. Springer, 2006, pp. 432–443.

[5] P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," in *CRYPTO 1999*, ser. Lecture Notes in Computer Science, vol. 1666. Springer-Verlag, 1999, pp. 388–397.

[6] E. Brier, C. Clavier, and F. Olivier, "Correlation Power Analysis with a Leakage Model," in *CHES 2004*, ser. Lecture Notes in Computer Science, vol. 3156. Springer, 2004, pp. 16–29.

[7] P. Gauravaram and K. Okeya, "Side Channel Analysis of Some Hash Based MACs: A Response to SHA-3 Requirements," in *ICICS 2008*, ser. Lecture Notes in Computer Science, vol. 5308. Springer, 2008, pp. 111–127.

[8] O. Benoît and T. Peyrin, "Side-Channel Analysis of Six SHA-3 Candidates," in *CHES 2010*, ser. Lecture Notes in Computer Science, vol. 6225. Springer, 2010, pp. 140–157.

[9] M. Zohner, M. Kasper, and M. Stöttinger, "Side Channel Evaluation of SHA-3 Candidates," First International Workshop on Trustworthy Embedded Devices TRUSTED, 2011.

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

[11] 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.

[12] K. Yasuda, ""Sandwich" Is Indeed Secure: How to Authenticate a Message with Just One Hashing," in *ACISP 2007*, ser. Lecture Notes in Computer Science, vol. 4586. Springer, 2007, pp. 355–369.

[13] T. S. Messerges, E. A. Dabbish, and R. H. Sloan, "Investigations of Power Analysis Attacks on Smartcards," in *USENIX Workshop on Smartcard Technology*, 1999, pp. 151–162.

[14] L. Goubin and J. Patarin, "DES and Differential Power Analysis (The "Duplication" Method)," in *CHES 1999*, ser. Lecture Notes in Computer Science, vol. 1717. Springer, 1999, pp. 158–172.

[15] L. Goubin, "A Sound Method for Switching between Boolean and Arithmetic Masking," in *CHES 2001*, ser. Lecture Notes in Computer Science, vol. 2162. Springer, 2001, pp. 3–15.

[16] J.-S. Coron and A. Tchulkine, "A New Algorithm for Switching from Arithmetic to Boolean Masking," in *CHES 2003*, ser. Lecture Notes in Computer Science, vol. 2779. Springer, 2003, pp. 89–97.