Differential Power Analysis of HMAC SHA-2 in the Hamming Weight Model

Sonia Belaïd, Luk Bettale, Emmanuelle Dottax, Laurie Genelle, Franck Rondepierre

Differential Power Analysis of HMAC SHA-2 in the Hamming Weight Model

Sonia Belaïd[1,2] *, Luk Bettale[3], Emmanuelle Dottax[3], Laurie Genelle[3] and Franck Rondepierre[3]

[1] *École Normale Supérieure, 45 rue d'Ulm, 75005 Paris, France*

[2] *Thales Communications & Security, 4 Avenue des Louvresses, 92230 Gennevilliers, France*

[3] *Oberthur Technologies, Cryptography Group, 420 rue d'Estienne d'Orves, 92700 Colombes, France*

sonia.belaid@ens.fr, {l.bettale, e.dottax, l.genelle, f.rondepierre}@oberthur.com

Keywords: Side Channel Analysis : Differential Power Analysis : Hamming Weight : HMAC : SHA-2

Abstract: As any algorithm manipulating secret data, HMAC is potentially vulnerable to side channel attacks. In 2007, McEvoy et al. proposed a differential power analysis attack against HMAC instantiated with hash functions from the SHA-2 family. Their attack works in the Hamming distance leakage model and makes strong assumptions on the target implementation. In this paper, we present an attack on HMAC SHA-2 in the Hamming weight leakage model, which advantageously can be used when no information is available on the targeted implementation. Furthermore, our attack can be adapted to the Hamming distance model with weaker assumptions on the implementation. We show the feasibility of our attack on simulations, and we study its overall cost and success rate. We also provide an evaluation of the performance overhead induced by the countermeasures necessary to avoid the attack.

1 INTRODUCTION

With the expansion of internet communications, online transactions and the transfer of confidential data in general, ensuring the integrity and the authenticity of transmitted information is a prime necessity. To this end, a *Message Authentication Code* (MAC) is generally used. A MAC algorithm accepts as input a secret key – shared between senders and receivers – and an arbitrarily long message. The output is a short bit-string which is jointly transmitted with the message. It allows the receiver to verify that the message has not been altered by an attacker.

Several MAC constructions exist, and the most common ones are based on block-ciphers or on hash functions. Among the hash-based MAC algorithms, HMAC [Bellare et al., 1996] is the most widely used. Today it is a standardized algorithm [FIPS 198-1, 2008] and it is used by several protocols running on embedded devices [Haverinen and Salowey, 2006, Arkko and Haverinen, 2006]. The use of HMAC in such a context leads the research community to study its vulnerability against *Side Channel Analysis* (SCA) attacks. Those attacks take advantage of statistical dependencies that exist between a *physical leakage* (e.g., the power consumption, the electromagnetic emanations) produced during the execution of a cryptographic algorithm and the intermediate values manipulated. In the family of side channel analyses, *Differential Power Analysis* (DPA) is of particular interest [Kocher et al., 1999]. The principle is the following. The attacker executes the cryptographic algorithm several times with different inputs and gets a set of power consumption traces, each trace being associated to one value known by the attacker. At some points in the algorithm execution, *sensitive variables* are manipulated, i.e., variables that can be expressed as a function of the secret key and the known value. These sensitive values are targeted as follows: the attacker makes hypotheses about the secret key and predicts the sensitive values and the corresponding leakages. Then, a statistical tool is used to compute the correlation between these predictions and the acquired power consumption traces. The obtained correlation values allow the attacker to (in)validate some hypotheses. In order to map the hypothetical sensitive value towards an estimated leakage, a model function must be chosen. The *Hamming Distance* (HD) and the *Hamming Weight* (HW) models are the most commonly used by attackers to simulate the power con-

*This work was essentially done while this author was a member of the Cryptography Group of Oberthur Technologies.

sumption of an embedded device. In the HW model, the leakage is assumed to rely on the number of bits that are set in the handled data. It is considered as a special case of the HD model, which assumes that the leakage depends on the bits switching from one state to the next one. The latter is usually considered to better integrate the behavior of CMOS circuits, however it requires significant knowledge of the implementation. As for the HW model, it can always be used and gives valid results for a large number of devices [Kocher et al., 1999, Messerges, 2000, Mangard et al., 2007].

Related Works. Several DPA scenarios have been proposed in the literature to attack the HMAC algorithm. Okeya et al. addressed in several papers [Okeya, 2006, Gauravaram and Okeya, 2007, Gauravaram and Okeya, 2008] the question of protecting HMAC against DPA. They focused their study on block-cipher based hash functions. As well, [Zhang and Shi, 2011] dealt with HMAC based on Whirlpool. In [Lemke et al., 2004], Lemke et al. described a theoretical attack on HMAC based on the hash functions RIPEMD-160 and SHA-1 in the HW model. McEvoy et al. [McEvoy et al., 2008] proposed an attack against HMAC based on SHA-2 functions. They chose the HD model to characterize the physical leakage of the device. The paper [Fouque et al., 2009] presents a template attack on HMAC SHA-1, which implies a more powerful adversary than DPA [Chari et al., 2002]. More recently, DPA on keyed versions of KECCAK have been explored in [Zohner et al., 2012, Bertoni et al., 2013].

Contributions. In this paper, we improve the state of the art on the security of HMAC against DPA by proposing an attack in the HW model. Contrary to [McEvoy et al., 2008], our attack can be used even when no information about the HMAC implementation is available. Moreover, our attack can easily be adapted to the HD model, and it turns out that the resulting attack requires weaker assumptions on the HMAC implementation than the ones made in [McEvoy et al., 2008]. Indeed, the attack by McEvoy et al. relies on a constraining HMAC implementation, which reduces the scope of their attack. We also study the cost and the success rate of the attack, that leads to the first complete study of a full DPA attack complexity on HMAC. We focus our study on HMAC based on SHA-256, however our work can be straightforwardly adapted to all SHA-2 family functions, and to RIPEMD-160, MD5 and SHA-1 with small modifications.

Paper Organisation. The rest of the paper is organized as follows. Section 2 introduces the necessary background on HMAC and SHA-256 algorithms. Section 3 discusses the interest of our attack and describes the details. Section 4 exhibits the results of simulations and evaluates the efficiency of the new attack on unprotected implementations. Eventually, Sect. 5 deals with the protections required to secure a HMAC implementation against our attack, and notably it evaluates the impact on performances.

2 TECHNICAL BACKGROUND

2.1 The HMAC Construction

The HMAC cryptographic algorithm involves a hash function H in combination with a secret key k. According to [FIPS 198-1, 2008], it is defined as follows:

$$\begin{array}{rcl} \mathrm{HMAC}_k: & & \\ \{0,1\}^* & \longrightarrow & \{0,1\}^h \\ m & \longmapsto & \mathrm{H}\left((k \oplus opad) \parallel \mathrm{H}\left((k \oplus ipad) \parallel m\right)\right), \end{array}$$

where $\oplus$ denotes the bitwise *exclusive or*, $\parallel$ denotes the concatenation, and $opad$ and $ipad$ are two public fixed constant. We call *inner hash* the first hash computation $\mathrm{H}\left((k \oplus ipad) \parallel m\right)$ and the second one is referred to as the *outer hash*.

In this paper, we focus on HMAC instantiated with a hash function H based on the Merkle-Damgård construction [Merkle, 1989, Damgård, 1989] (MD5, SHA-1 and SHA-2 are among the most widely used). An overview of this construction is given in Fig. 1. The input message m is first padded using a specific procedure to obtain N blocks of bit-length n denoted by $m_1, \ldots, m_N$. Then each block m_i is processed with a h-bit chaining value CV_{i-1} through a one-way compression function F that outputs a new h-bit chaining value CV_i. The chaining value CV_0, also denoted by k_1, is fixed and depends only on the secret key k. It is computed as $\mathrm{F}(\mathrm{IV}, k \oplus ipad)$, with IV being the public *Initial Value* of the hash function. The final chaining value CV_N, also denoted by z, is the input of the outer hash. It is processed with a second fixed key-dependent value $k_0 = \mathrm{F}(\mathrm{IV}, k \oplus opad)$ in the last call of the compression function that outputs the MAC. So we rewrite the HMAC procedure as follows:

$$\begin{aligned} &\mathrm{HMAC}_k(m) = \\ &\quad \mathrm{F}\left(k_0, \mathrm{F}\left(\ldots \mathrm{F}\left(\mathrm{F}\left(k_1, m_1\right), m_2\right), \ldots, m_N\right) \parallel pad\right), \end{aligned}$$

where pad is the bit-string used to pad the input of the outer hash. For the sake of simplicity and without loss of generality, we omit this value in the following.

In the rest of the paper we make our analysis on the HMAC algorithm based on SHA-256. We assume F to be the SHA-256 compression function. A brief description is given in the next section.

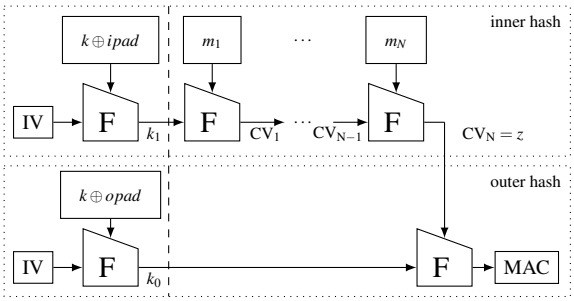

Figure 1: HMAC using a Merkle-Damgård hash function

2.2 The SHA-256 Compression Function

The SHA-256 compression function F is described in Alg. 1. It accepts as input a 512-bit message block M and a 256-bit chaining value V (i.e., parameters n and h in Sect. 2.1 equal 512 and 256 respectively). The function iterates 64 times the same round transformation on an internal state. The state is represented by eight 32-bit words A,B,C,D,E,F,G and H initially filled with $V = (V_1,\ldots,V_8)$. The round is a composition of 32-bit modular additions, denoted by $\boxplus$, with boolean operations which are defined on 32-bit words u, v and w as follows:

$$\mathrm{Ch}(u,v,w) = (u \wedge v) \oplus (\neg u \wedge w),$$
$$\mathrm{Maj}(u,v,w) = (u \wedge v) \oplus (u \wedge w) \oplus (v \wedge w),$$
$$\Sigma_0(u) = (u \ggg 2) \oplus (u \ggg 13) \oplus (u \ggg 22),$$
$$\Sigma_1(u) = (u \ggg 6) \oplus (u \ggg 11) \oplus (u \ggg 25),$$

where $\wedge$ denotes the bitwise *and*, $\neg$ denotes the bitwise complement and $x \ggg j$ denotes a rotation of j bits to the right on x.

The message expansion splits the message block M into 32-bit words $M_1,\ldots,M_{16}$, and expands it into 64 words W_t by using the following additional 32-bit words operations:

$$\sigma_0(u) = (u \ggg 7) \oplus (u \ggg 18) \oplus (u \gg 3),$$
$$\sigma_1(u) = (u \ggg 17) \oplus (u \ggg 19) \oplus (u \gg 10),$$

where $x \gg j$ denotes a shift of j bits to the right on x. In Alg. 1, the values $K_1,\ldots,K_{64}$ are public constants.

Algorithm 1 SHA-256 Compression Function

Inputs: the data block $M = (M_1,\ldots,M_{16})$, the chaining value $V = (V_1,\ldots,V_8)$
Output: the chaining value $\mathrm{F}(V,M)$

1: $(W_1,\ldots,W_{16}) \leftarrow (M_1,\ldots,M_{16})$
2: **for** $t = 17$ **to** 64 **do** ▷ Message Expansion
3: $W_t \leftarrow \sigma_1(W_{t-2}) \boxplus W_{t-7} \boxplus \sigma_0(W_{t-15}) \boxplus W_{t-16}$
4: **end for**
5: $(A,B,C,D,E,F,G,H) \leftarrow (V_1,\ldots,V_8)$
6: **for** $t = 1$ **to** 64 **do** ▷ Main Loop
7: $T_1 \leftarrow H \boxplus \Sigma_1(E) \boxplus \mathrm{Ch}(E,F,G) \boxplus K_t \boxplus W_t$
8: $T_2 \leftarrow \Sigma_0(A) \boxplus \mathrm{Maj}(A,B,C)$
9: $H \leftarrow G$
10: $G \leftarrow F$
11: $F \leftarrow E$
12: $E \leftarrow D \boxplus T_1$
13: $D \leftarrow C$
14: $C \leftarrow B$
15: $B \leftarrow A$
16: $A \leftarrow T_1 \boxplus T_2$
17: **end for**
18: **return** $(V_1 \boxplus A,\ \ldots,\ V_8 \boxplus H)$ ▷ Final Addition

3 DPA ON HMAC SHA-256

3.1 Related Work and Contribution

In [McEvoy et al., 2008], the authors propose to attack the SHA-256 compression function to recover k_0 and k_1. The authors mount their attack in the HD leakage model, and they assume to have knowledge (only) of the input messages. They consider an implementation that strictly follows Alg. 1, and in particular they make the following assumptions. Firstly, the variables $A,B,\ldots,H$ are initialized with the input chaining value and T_1 is initialized with an unknown but constant value. Secondly, each one of the variables $T_1,T_2,A,B,\ldots,H$ is updated with its value at the next round. It means that for each of these variables, the HD between its value at round $t-1$ and its value at round t is leaked at each round t, for $t = 1,\ldots,64$. Under these assumptions, the authors present an attack wich consists in a succession of DPAs. Each one allows the attacker to recover either a part of the secret key or an intermediate result, and these results are re-used in the following DPAs to recover the remaining secrets. It is worth noticing that these assumptions are quite strong and could prevent applying the attack on some implementations. For instance, a software implementation would probably avoid updating registers value (steps 9 to 16 of Alg. 1) and rather choose to directly update the pointers, which would clearly be more efficient.

In this paper, we propose an attack on the compression function that targets different steps in the algorithm compared to [McEvoy et al., 2008]. This new method brings two main advantages. First, our new attack benefits from the feature to work in the HW model in which the power consumption is assumed to be proportional to the number of non-zero bits of the

processed values. Therefore our proposal can be applied on devices that leak in this model, and also when the attacker has no information about the implementation, as stated in [Mangard et al., 2007]. Secondly, we show in Sect. 3.3 that our proposal can also be turned into an attack in the HD model but with less restrictive assumptions than [McEvoy et al., 2008], which advantageously extends the scope of the attack.

3.2 New Attack in the Hamming Weight Model

To forge MACs for arbitrary messages, the attacker needs either to recover the secret key k or both values k_0 and k_1. As seen in Fig. 1, the attacker cannot target directly the secret key k since it is never combined with variable and known data. On the contrary, k_0 and k_1 may potentially be recovered by the attacker. In the following, we define the three paths the attacker can follow to recover k_1 and k_0 (they are shown by Fig. 2). Then, Sect. 3.2.1, 3.2.2 and 3.2.3 give the detailed steps of the attacks following respectively Path 1, 2 and 3.

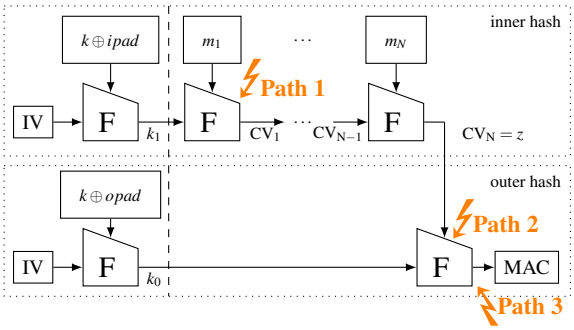

Figure 2: Attack paths on HMAC

As already noted, the value k_1 may be obtained when it is combined with the known and variable data m_1 in the compression function. This attack path is referred to as Path 1.

Definition 1 (Path 1: Inner hash - DPA with known input.)**.** *The attacker targets the compression function whose input is the first message block m_1 to recover the secret value k_1.*

Once k_1 is known, the attacker is able to compute the inner hash result $z = H(k_1||m)$ for all input messages m. She can thus mount a DPA on the outer hash compression function execution whose input is z to recover the constant value k_0. This path is denoted by Path 2.

Definition 2 (Path 2: Outer hash - DPA with known input.)**.** *The attacker targets the last call of the compression function whose input is the known and variable value z.*

Another way for the attacker to obtain the secret value k_0 is to target the last call of the compression function focussing on the MAC value which is known and variable. We refer to this attack path as Path 3.

Definition 3 (Path 3: Outer hash - DPA with known output.)**.** *The attacker targets the last compression function execution whose output is the known and variable MAC.*

3.2.1 Path 1

We depict here the attack following Path 1, i.e., on the computation $F(k_1, m_1)$. In this context, the attacker aims at recovering the secret value k_1. We completely develop this attack in Table 1. The notation $X^{(i)}$ refers to a given intermediate variable X computed at round i. Variables denoted by $X^{(0)}$ correspond to the input chaining value of the compression function. For the sake of simplicity, $\delta^{(i)}$ denotes the sum $H^{(i)} \boxplus \Sigma_1\left(E^{(i)}\right) \boxplus \mathrm{Ch}\left(E^{(i)}, F^{(i)}, G^{(i)}\right) \boxplus K_{i+1}$. Eventually, $\widehat{X}$ denotes a variable controlled by the attacker, meaning that she can predict its value when the message changes.

Each line of the table describes a DPA attack. The column **Hyp** indicates the secret value which is the target of the attack **Attack** in the operation **Targeted Operation**. In each targeted operation, the hat indicates the variable that is controlled (modified) by the attacker. The column **Result** lists the useful variables on which the attacker gains control after the attack (it includes, but is not limited to, the target secret variables). Eventually, the double line separates the attacks executed in Round 1 from the ones processed in Round 2.

The attacker progresses line after line and finally recovers the following parts of the secret: $A^{(0)}, B^{(0)}, D^{(0)}, E^{(0)}, F^{(0)}, G^{(0)}$. The last remaining parts $H^{(0)}$ and $C^{(0)}$ can be recovered by making substitutions in Alg. 1: in Step 7 of round 1, where $H^{(0)}$ is the only unknown variable, and similarly in Step 8 of round 1 where $C^{(0)}$ is the only unknown variable.

Remark. DPA 8 involves the message block W_2. The attacker has two possibilities to mount this attack:

1. She can fix the first message block W_1 and thus makes hypotheses on the whole constant sum $\delta^{(1)}$, while modifying W_2. She obtains the value of $\delta^{(1)}$ and deduces the secret $H^{(1)}$ from the knowledge of the other values.

Table 1: DPA attack on SHA-256 compression function using HW leakage model

Attack	Targeted Operation	Hyp	Result
DPA 1	$T_1^{(1)} \leftarrow \delta^{(0)} \boxplus \widehat{W}_1$	$\delta^{(0)}$	$\widehat{T}_1^{(1)}, \delta^{(0)}$
DPA 2	$E^{(1)} \leftarrow D^{(0)} \boxplus \widehat{T}_1^{(1)}$	$D^{(0)}$	$\widehat{E}^{(1)}$, $\mathbf{D^{(0)}}$
DPA 3	$A^{(1)} \leftarrow \widehat{T}_1^{(1)} \boxplus T_2^{(1)}$	$T_2^{(1)}$	$\widehat{A}^{(1)}, T_2^{(1)}$
DPA 4	$\widehat{E}^{(1)} \wedge F^{(1)}$ in Ch	$F^{(1)}$	$F^{(1)} = \mathbf{E^{(0)}}$
DPA 5	$\overline{\widehat{E}^{(1)}} \wedge G^{(1)}$ in Ch	$G^{(1)}$	$G^{(1)} = \mathbf{F^{(0)}}$
DPA 6	$\widehat{A}^{(1)} \wedge B^{(1)}$ in Maj	$B^{(1)}$	$B^{(1)} = \mathbf{A^{(0)}}$
DPA 7	$\widehat{A}^{(1)} \wedge C^{(1)}$ in Maj	$C^{(1)}$	$C^{(1)} = \mathbf{B^{(0)}}$
DPA 8	$T_1^{(2)} \leftarrow H^{(1)} \boxplus \Sigma_1\left(E^{(1)}\right) \boxplus \mathrm{Ch} \boxplus K_2 \boxplus \widehat{W}_2$	$H^{(1)}$	$H^{(1)} = \mathbf{G^{(0)}}$

2. W_1 is not fixed, but rather changes together with W_2. She then considers the sum $\Sigma_1\left(E^{(1)}\right) \boxplus \mathrm{Ch}\left(E^{(1)}, F^{(1)}, G^{(1)}\right) \boxplus K_2 \boxplus W_2$ as the variable to mount the DPA. Knowing the values taken by the variable and making hypotheses on the secret $H^{(1)}$, she obtains as well the targeted value.

Both methods require the same number of traces and are applicable with respect to the attack model. However, note that fixing W_1 may be more convenient since there is no need to compute $E^{(1)}$ for each execution.

The combination of these eight DPAs allows an attacker to recover the input chaining value k_1 from the observation of the first two rounds of F only.

3.2.2 Path 2

The attack related to Path 2 to recover k_0 follows the same outline as the one associated to Path 1. Indeed, it targets the computation $F(k_0, z)$ in the outer hash, whose input is z. However, in this context the value z is known for any input message but not chosen. As a consequence, the attacker cannot easily fix the first message block and would probably choose the second alternative to mount DPA 8 in Table 1.

3.2.3 Path 3

The attack related to Path 3 targets the same compression function execution as Path 2. It also aims at recovering the same secret value k_0 but focuses on the output of the compression function. Indeed in the HMAC algorithm, the last call to the compression function outputs the MAC value R. This final value is obtained by performing a final addition between the secret chaining input $V = k_0$ and the output of a 64-round process. Thus we have:

$$A^{(64)} = R_1 \boxminus V_1 ,$$
$$B^{(64)} = R_2 \boxminus V_2 ,$$
$$\ldots$$
$$H^{(64)} = R_8 \boxminus V_8 ,$$

where $\boxminus$ is the modular subtraction on 32 bits. In these final operations, the $(R_i)_{1 \leqslant i \leqslant 8}$ are known and variable and the $(V_i)_{1 \leqslant i \leqslant 8}$ are constant parts of the secret k_0, thus the values $A^{(64)}, \ldots, H^{(64)}$ are sensitive. Eight DPA attacks can thus be mounted to recover the eight 32-bit parts $(V_i)_{1 \leqslant i \leqslant 8}$ of the secret k_0.

3.2.4 Full Attack

To conclude, the attacker can follow either Paths 1 and 2 or Paths 1 and 3 to recover the secret values required to forge MACs. In both cases, she needs to mount sixteen DPAs on 32-bit words. As mentioned above, the attack can be generalized on HMAC instantiated with any of the SHA-2 family hash function with few adaptations. Indeed, the other SHA-2 hash functions differ either in the size of the internal variables (32 bits or 64 bits), or in the length of the final output. For the DPAs to be computationally practical when mounted on 32-bit or 64-bit values, one can use partial DPAs [Lemke et al., 2004] as explained in Sect. 4.1.2. For HMAC implementations whose final output is truncated, the attacker cannot directly follow Path 3 to recover k_0 but is still able to use Path 2.

3.3 New Attack in the Hamming Distance Model

If the device attacked is known to leak in the HD model, our proposal can be adjusted into an attack in the HD leakage model. To do so, we make the assumption that the variables $T_1, T_2, A, B, \ldots, H$ are initialized with unknown but constant values. Then, the DPA attack can be mounted provided that we make additional hypotheses on these initial values. This implies making 64-bit hypotheses, which can be handled using partial DPA as done in [McEvoy et al., 2008].

Our new attack benefits from the feature to require less restrictive assumptions on the implementation. Indeed, contrary to [McEvoy et al., 2008], our proposal does not expect the initial values to be equal to

the input chaining value. It also removes the requirement for the variables $T_1, T_2, A, B, \ldots, H$ to be updated with their next values.

Eventually our attack in the HD model is as efficient as the attack in [McEvoy et al., 2008] in terms of number of DPAs. However, the scope of our proposal is definitely larger than the existing attack.

4 ATTACK COST EVALUATION

In this section, we focus on evaluating the cost of the attack paths described in Sect. 3. To achieve it, first we give some background on DPA and in particular on so-called partial DPA [Lemke et al., 2004, Tunstall et al., 2007]. Then we explain how to apply it in the particular case of an unprotected HMAC implementation. Eventually we give an overview of the total cost of the full attack to retrieve the two secret keys k_0 and k_1.

4.1 DPA Background

4.1.1 DPA Process

As already mentioned, a DPA exploits the statistical dependency between the secret key and physical leakages. This dependency results from the manipulation of sensitive variables during the algorithm execution. Instead of observing values related to the whole secret key (for which brute-force attack is infeasible), DPA focuses on sensitive values linked to a relatively small part K of the secret. The size of K depends on the processed algorithm and the chip architecture. For instance, on 32-bit processors, data are manipulated by 32-bit chunks. We express a sensitive variable as a function g of K and a known value M. An attacker can then test hypotheses $\widehat{K}$ by comparing the predicted leakage to the measured leakage. We sum-up hereafter the different steps:

1. Measure the leakage $(l_i)_i$ produced by the N calculations of $g(K, M_i)$ using a sample $(M_i)_i$ of N values. If $\mathcal{L}$ denotes the leakage of a physical device then $l_i = \mathcal{L} \circ g(K, M_i)$.
2. Select a prediction function P that approximates the leakage function $\mathcal{L}$.
3. For each hypothesis $\widehat{K}$, compute the correlation between the predicted leakage $(P \circ g(\widehat{K}, M_i))_i$ and the observed leakage $(l_i)_i$. In this paper we evaluate the correlation by using Pearson's coefficient [Brier et al., 2004].
4. The hypothesis $\widehat{K}$ maximizing this correlation is assumed to be the secret part K.

4.1.2 Partial DPA

In practice, the value K may remain too large to mount a DPA. However, for some functions g, it is possible to predict some bits of $g(K, M_i)$ with assumptions on only few bits (say ℓ) of K. This property enables to use partial DPA in order to guess the whole secret K by blocks of ℓ bits. If we assume that K is split such that $K = \sum_j K_j \cdot 2^{j\ell}$, then a partial DPA follows the same steps as described above but it targets the processing of g related to the j^{th} ℓ-bit block of K (denoted $g_\ell(K, M_i, j)$ for a given message M_i). We denote by partial DPA the process of recovering the whole secret K with several ℓ-bit DPAs.

Partial DPA is a useful tool to limit the attack cost. Indeed, if we denote by $O(N)$ the number of operations required to compute one correlation coefficient for N messages/leakages, a classical DPA on N messages needs $2^w O(N)$ operations to recover a w-bit secret, whereas a partial DPA requires a total of $\left(\frac{w}{\ell} \cdot 2^\ell\right) O(N)$ operations ($\frac{w}{\ell}$ ℓ-bit DPA). The main drawback is that the measured leakage depends on w bits, but the correlation is performed only on ℓ bits. This means that the $w - \ell$ other bits are assimilated to noise in the DPA correlation computation. More precisely if ρ is the Pearson correlation coefficient of the leakage of the ℓ-bit useful data taken separately, the total correlation coefficient of the w-bit data is in average $\sqrt{\frac{\ell}{w}}\,\rho$ (see for instance [Brier et al., 2004, Tunstall et al., 2007]).

4.2 Partial DPAs on HMAC Operations

In the following, we consider the leakage $\mathcal{L}$ resulting from the manipulation of a sensitive variable x as the sum of two terms: the information expressed as the Hamming weight of x and an independent noise denoted by ε. We assume that the noise follows a Gaussian distribution with a null mean and a standard deviation σ (denoted by $\mathcal{N}(0, \sigma)$):

$$\mathcal{L}(x) = \mathrm{HW}(x) + \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, \sigma) \ .$$

As already seen in Table 1, only two elementary operations are involved: the 32-bit modular addition ($\boxplus$) and the 32-bit bitwise *and* ($\wedge$). Hence the function g introduced in the previous section is either the primitive operation $\boxplus$ or $\wedge$, and the length of the data equals 32 bits. Let us introduce the prediction function used to approximate the leakage related to the j^{th} ℓ-bit block. In the case of modular addition, the value of the first $j-1$ blocks has an impact on the value of the j^{th} block because of the carry propagation. Thus, when starting the attack from the least significant block, the first $j-1$ guessed blocks have to be

reused to approximate the leakage of the j^{th} block. A similar process can be used for the bitwise *and*. Our prediction function for both operations is

$$P(g_\ell(\widehat{K}, M_i, j)) = \text{HW}\left(g\left(\widehat{K_j}\cdot 2^{j\ell} + \sum_{k=0}^{j-1} K_k \cdot 2^{k\ell}, M_i\right) \bmod 2^{(j+1)\cdot\ell}\right),$$

for each j, $0 \leqslant j < \frac{32}{\ell}$. Eventually, we assume that HMAC is processed on a 32-bit processor (i.e., nowadays the largest size for embedded devices). This is the worst case for an attacker to mount partial DPAs as the leakage of $32-\ell$ bits will be assimilated to noise.

It has to be noted that with modular addition, it is not possible to perform 1-bit partial DPA. Indeed, if $\ell = 1$, a modular addition is an *exclusive or* operation. The hypothesis $K = 0$ cannot be distinguished from $K = 1$ (see for instance [Lemke et al., 2004]). As for the bitwise *and*, the hypothesis $K = 0$ does not bring any information. As $M_i \wedge K = 0$ for any message M_i the leakage is just a measurement of noise. While the zero key case is very unlikely for large values of ℓ, it becomes a problem for small values. It is even impossible to perform a partial DPA on bitwise *and* for $\ell = 1$. In what follows, we consider only partial DPA on $\ell > 1$ bits.

4.2.1 Simulations

Now let us compare the results of the partial DPA applied on the two operations. We consider partial DPA on 8-bit words (i.e., $\ell = 8$). We depict in Fig. 3 (resp. Fig. 4) the correlation progression according to the number of messages used to attack the modular addition (resp. bitwise *and*). We say that the DPA converges when one hypothesis remains above the other as the number of messages grows. Each figure is composed of $\frac{32}{\ell} = 4$ graphs from top to bottom. The graphs on the top show the correlation for the eight least significant bits. The most correlated hypothesis is then reused in the guess of the eight next bits, and so on until the eight most significant bits (bottom graphs) are guessed.

As expected, the value of the correlation is lesser for the least significant bits because the other bits are not used in the leakage prediction function g_ℓ. When comparing attacks on both operations, we observe that more messages are required in the bitwise *and* case for the DPA to converge. Moreover we notice that the correlation is lower and that the correlation of the second most correlated hypothesis is closer to the correct one. This can be explained by the characteristic of the bitwise *and*. For instance, consider one key hypothesis K_1 and the key hypothesis $K_2 = K_1 \oplus 1$. Then, for all message M_i such that

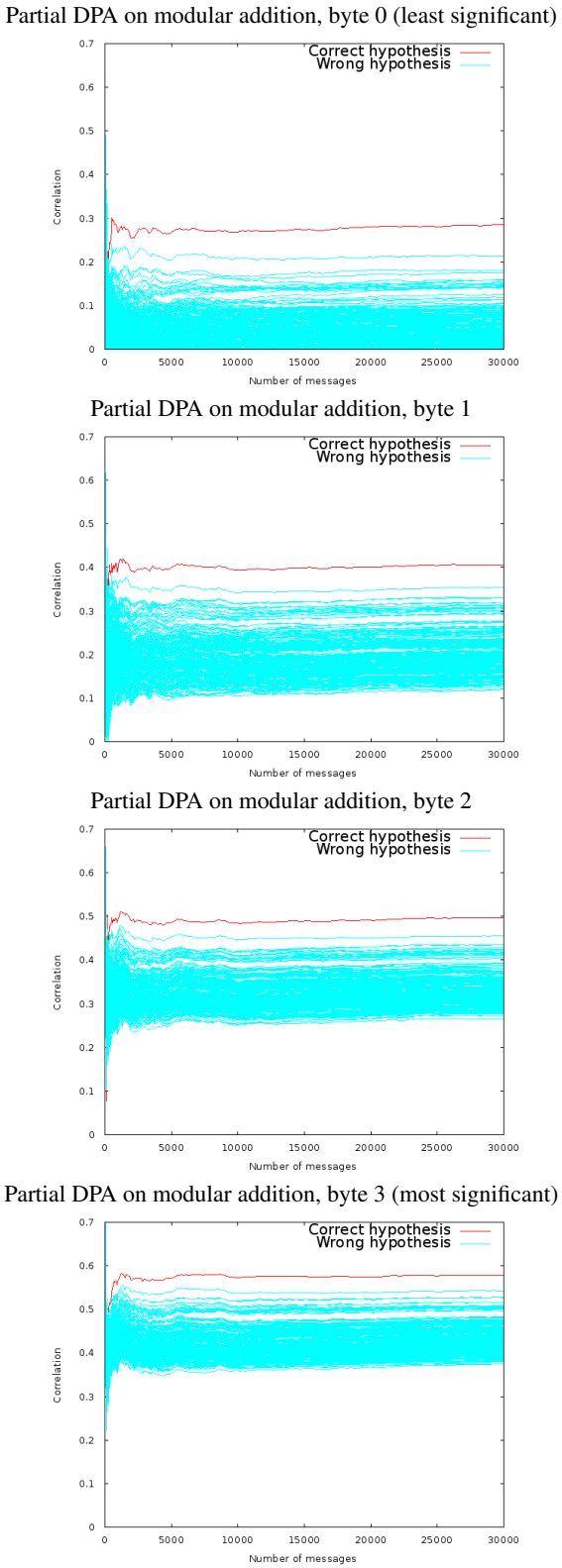

Figure 3: Attack on 32-bit addition with $\ell = 8$ bits partial DPA based on simulated curves with $\sigma = 4$. Top-most graph shows the partial DPA on the least significant byte.

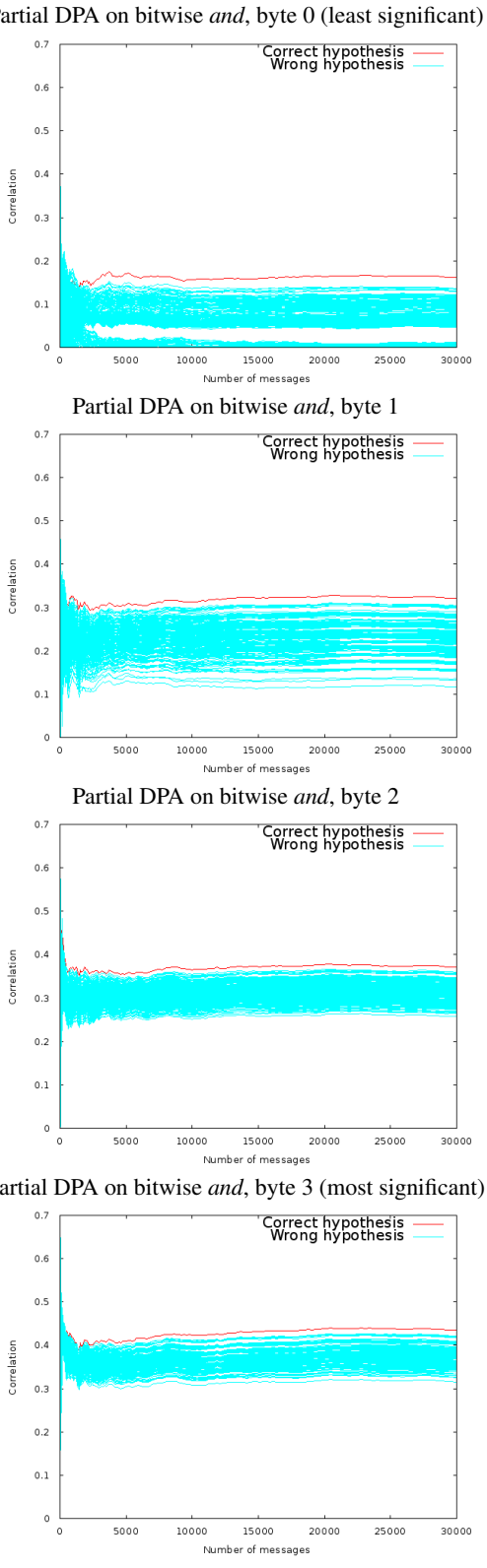

Figure 4: Attack on 32-bit logical AND with $\ell = 8$ bits partial DPA based on simulated curves with $\sigma = 4$. Top-most graph shows the partial DPA on the least significant byte.

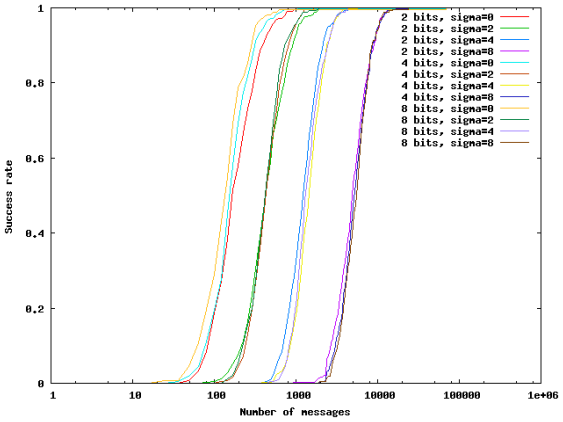

Figure 5: Success rate of partial DPA when attacking a bitwise *and* when the special case 0 is not taken into account. The success rate is the same for any value ℓ.

$M_i \wedge 1 = 0$, $HW(K_1 \wedge M_i) = HW(K_2 \wedge M_i)$, and the difference will be only 1 for the other messages. Both hypotheses have similar correlation coefficients.

4.3 Full HMAC Attack

We focus on the cost and the success rate of an attack on the full HMAC using ℓ-bit partial DPA. The success rate is the probability that the best key hypothesis revealed by the partial DPA is the correct key. We denote by $\mathcal{P}_{\boxplus,\ell}(N)$ (resp. $\mathcal{P}_{\wedge,\ell}(N)$) the success rate of an ℓ-bit DPA for $\boxplus$ (resp. $\wedge$) with N messages. Note that the case of full length DPA is covered by taking $\ell = 32$. We simulated this success rate on $\boxplus$ and $\wedge$ with different noises $\sigma \in \{0,2,4,8\}$ for an ℓ-bit partial DPA with $\ell \in \{4,8\}$ (Fig. 5 and 6). Note that these success rates can be computed on a copy of the target device independently of the HMAC algorithm. These success rates will serve to compute the success rate of the whole attack. It can be observed that the curves in Fig. 5 are gathered by noise level. This means that the success rate seems not to depend on the size ℓ of the partial DPA, but only on the amount of noise and the number of messages. In Fig. 5 the special case of the 0 key has not been taken into account. We assume that the partial DPA fails when at least one of the ℓ-bit chunks of the key is zero. The success rate has then to be multiplied by the probability of not having a 0 subkey which is $\left(\frac{2^\ell-1}{2^\ell}\right)^{\frac{N}{\ell}}$. In regards of our experimental results, we can write

$$\mathcal{P}_{\wedge,\ell}(N) = \left(\frac{2^\ell - 1}{2^\ell}\right)^{\frac{32}{\ell}} \cdot \mathcal{P}_{\wedge}(N) \quad , \qquad (1)$$

where $\mathcal{P}_{\wedge}(N)$ denotes the success rate of a partial DPA with N messages for any value ℓ without 0 subkey.

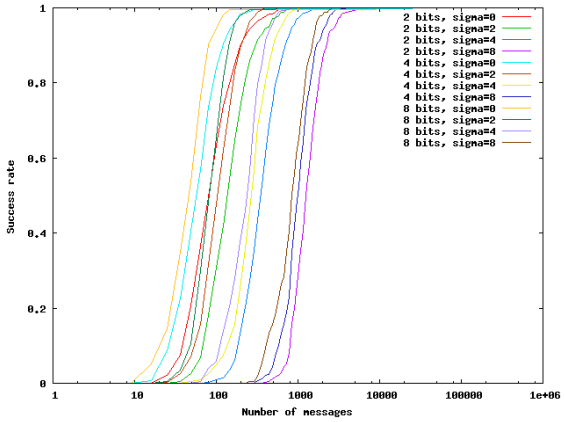

Figure 6: Success rate of partial DPA when attacking a modular addition. The success rate is always greater than the one of bitwise *and*.

Moreover, for a same number of messages, the success rate of modular addition is always greater than the one of the bitwise *and* for any value of ℓ. In our success rate evaluation, we may then use the following assumption:

$$\mathcal{P}_{\boxplus,\ell}(N) = \mathcal{P}_{\boxplus}(N) \geqslant \mathcal{P}_{\wedge}(N) \quad . \tag{2}$$

These values are enough to determine the success rate of the whole attack presented in Sect. 3.

4.3.1 Attack cost and success rate

The only way to verify that a key is correct is to compute the HMAC of a message with the whole 2×256-bit keys k_0 and k_1. This means that one cannot tell that a partial DPA has failed until the whole keys have been tested. According to the attack paths described in Sect. 3, eight 32-bit DPAs are needed for each 256-bit. In Paths 1 and 2, four of them are DPAs on $\boxplus$ and the other four are DPAs on $\wedge$ operation. In Path 3, the eight DPAs are on $\boxplus$. All in all, when considering ℓ-bit partial DPA, one needs to perform a total of $\frac{512}{\ell}$ DPAs on ℓ bits. The total cost of an attack is $\left(\frac{512}{\ell} \cdot 2^{\ell}\right) O(N)$.

The total success probability of the whole attack is the combined success probability of each DPA taken independently. Using the success rates for partial DPA defined in (2) and (1), the success rate of Paths 1 or 2 is

$$\left(\mathcal{P}_{\boxplus}(N)\right)^4 \cdot \left(\left(\frac{2^{\ell}-1}{2^{\ell}}\right)^{\frac{32}{\ell}} \cdot \mathcal{P}_{\wedge}(N)\right)^4 \quad ,$$

and the success rate of Path 3 is

$$\left(\mathcal{P}_{\boxplus}(N)\right)^8 \quad .$$

Provided that she has access to the HMAC results, an attacker is then more likely to choose to attack HMAC using Path 1 to recover k_1 and then Path 3 rather than Path 2 to recover k_0 to maximize his success rate.

To minimize the complexity of the attack while keeping a good success rate, the attacker has to define the appropriate parameters ℓ and N. For instance we compute the total success rate of an attack (Path 1, Path 3) using $N = 2600$ messages, for $\ell = 8$-bit partial DPA in a setting with a standard noise of deviation $\sigma = 4$. According to our simulations, we obtain a success rate $\mathcal{P}_{\boxplus}(2600) = 1.00$ for modular addition and $\mathcal{P}_{\wedge}(2600) = 0.92$ for bitwise *and*. An attacker can mount an attack of complexity

$$\left(\overbrace{\underbrace{4 \cdot \frac{32}{8} \cdot 2^8}_{4\ \boxplus\ \text{operations}} + \underbrace{4 \cdot \frac{32}{8} \cdot 2^8}_{4\ \wedge\ \text{operations}}}^{\text{Path 1}} + \overbrace{\underbrace{8 \cdot \frac{32}{8} \cdot 2^8}_{4\ \boxplus\ \text{operations}}}^{\text{Path 3}} \right) O(N)$$

that is, 2^{14} correlation computations on N messages/leakages. The success probability of this attack is

$$\overbrace{\underbrace{1.00^4}_{4\ \boxplus\ \text{operations}} \times \underbrace{\left(\frac{2^8-1}{2^8}\right)^{\frac{32}{8}} \times 0.92^4}_{4\ \wedge\ \text{operations}}}^{\text{Path 1}} \times \overbrace{\underbrace{1.00^8}_{4\ \boxplus\ \text{ops.}}}^{\text{Path 3}} = 0.71 \, .$$

Our estimations allow an attacker to quickly evaluate the total cost and success rate of a full DPA attack on HMAC SHA-256 according to his settings and the number of available messages. Then he can adjust the parameter ℓ of the partial DPA to achieve the best trade-off between attack cost and success rate.

5 PROTECTED IMPLEMENTATION

In the previous section, it has been demonstrated that the theoretical attack paths presented in Sect. 3 are sound. In order to secure a HMAC implementation against this attack – and the one in [McEvoy et al., 2008] – adequate countermeasures must be applied. In software, the main techniques used to thwart such SCA are *masking* and *shuffling*, as well as combination of both [Rivain et al., 2009]. The principle is to inject some randomness in the algorithm execution, in order to reduce the amount of information that leaks on sensitive intermediate variables during the execution. In the rest of this section, we examine how to prevent attacks that use the previously presented

paths, and we provide an evaluation of the performance overhead, independently of the technique actually used to implement the countermeasures.

5.1 Preventing Paths 1 and 2

To mount an attack via Path 2, we recall that the attacker must be able to compute the intermediate value z for various messages. As shown in Sect. 3.2, this ability can be gained after recovering the value k_1 by an attack following Path 1, i.e., during the first compression function call $F(k_1, m_1)$ in the inner hash. However, preventing the recovery of k_1 is not sufficient to completely annihilate Path 2. Indeed, the knowledge of any of the chaining values CV_i in the inner hash still allows the attacker to compute the intermediate result z for fixed-prefix messages, and to mount an attack with Path 2. As every CV_i can be recovered by applying the attack following Path 1 to the corresponding compression function call, we deduce that every execution of the compression function in the inner hash has to be protected from attacks using Path 1. This is sufficient to prevent attacks via Path 2 as well.

Let us now see how to prevent the attack following Path 1. They rely on the observation of the first two rounds of the compression function, where the input message block m_i is manipulated together with the targeted secret values. It is thus necessary to protect the sensitive variables of the first two rounds. Furthermore, as we can see in Alg. 1, parts of the input message block are also involved in each one of the 64 rounds, via the message expansion output W_t. Thus, we have to check the feasibility of the attack on later rounds. We assume that the attacker adapts the attack described by Table 1 to rounds t and $t+1$, with $t \geq 2$. The first attack DPA 1 now relies on the variable W_t. The attacker can perform this attack and gain control on $T_1^{(t)}$ when the values involved in $\delta^{(t-1)}$ (namely $E^{(t-1)}, F^{(t-1)}, G^{(t-1)}$ and $H^{(t-1)}$) are constant. Afterwards, an adaptation of DPA 2 can be performed provided that $D^{(t-1)}$ is constant as well, and DPA 3 can be performed if $A^{(t-1)}, B^{(t-1)}$ and $C^{(t-1)}$ are constant. Remaining DPAs can then be performed, and the full internal state $(A^{(t-1)}, B^{(t-1)}, \ldots, H^{(t-1)})$ can be finally recovered. The attacker subsequently recovers previous states by inverting the round function, until she recovers the secret input chaining value $V = (A^{(0)}, B^{(0)}, \ldots, H^{(0)})$.

Coming back to the above described adaptation of DPA 1, the following two conditions must thus be fulfilled:

- values $E^{(t-1)}, F^{(t-1)}, G^{(t-1)}, H^{(t-1)}$ must be fixed,
- the value W_t must be variable.

To achieve the first condition, variables associated to the previous rounds $(W_1, \ldots, W_{t-1})$ must all be fixed as well. Yet, as soon as $t > 16$, the message expansion is such that constant values for $W_1, \ldots, W_{t-1}$ implies a constant value for W_t too, which contradicts the second condition. Hence, these two requirements can be fulfiled only for $t \leqslant 16$.

We conclude that the attack from Path 1 presented in Sect. 3.2.1 can be extended to any rounds among the first 16 rounds. Moreover, due to the structure of the compression function, some of the sensitive variables produced at round 16 remain available in rounds 17 to 20. Consequently, it is necessary to protect the sensitive variables until the 20th round.

5.2 Preventing Path 3

Section 3.2.3 describes an attack on the outer hash computation that targets the final addition of the last compression function call $F(k_0, z)$. We recall that the sensitive variables targeted by the attack are:

$$\begin{aligned} A^{(64)} &= R_1 \boxminus V_1, \\ B^{(64)} &= R_2 \boxminus V_2, \\ &\ldots \\ H^{(64)} &= R_8 \boxminus V_8, \end{aligned}$$

where the R_i's are known outputs, and the V_i's constitute the secret chaining input k_0. An attack can be mounted as soon as these sensitive values are manipulated. Rolling back the rounds of the compression function, we track these sensitive variables and present them in bold in Table 2. This shows that sensitive variables are produced from round 61 and that protection is required from round 61 to round 64, on top of the final addition.

Remark. One has to recall that for $1 \leqslant i \leqslant 64$:

$$\begin{aligned} T_1^{(i)} &\leftarrow H^{(i-1)} + \Sigma_1\left(E^{(i-1)}\right) \\ &\quad + \mathrm{Ch}\left(E^{(i-1)}, F^{(i-1)}, G^{(i-1)}\right) + K_i + W_i \ , \\ T_2^{(i)} &\leftarrow \Sigma_0\left(A^{i-1}\right) + \mathrm{Maj}\left(A^{(i-1)}, B^{(i-1)}, C^{(i-1)}\right) \ . \end{aligned}$$

Hence, even if the values of $T_1^{(i)}$ and $T_2^{(i)}$ are not sensitive, sensitive variables may be involved in their computations. For instance, the computation of $T_1^{(64)}$ involves the sensitive variables $E^{(63)}, F^{(63)}$, and $G^{(63)}$. A careful implementation is thus needed to avoid leakage from intermediate results.

Table 2: Sensitive variables in last rounds

Round 64	**Round 63**
$\boldsymbol{A}^{(64)} \leftarrow T_1^{(64)} \boxplus T_2^{(64)}$	$\boldsymbol{A}^{(63)} \leftarrow T_1^{(63)} \boxplus T_2^{(63)}$
$\boldsymbol{B}^{(64)} \leftarrow \boldsymbol{A}^{(63)}$	$\boldsymbol{B}^{(63)} \leftarrow \boldsymbol{A}^{(62)}$
$\boldsymbol{C}^{(64)} \leftarrow \boldsymbol{B}^{(63)}$	$\boldsymbol{C}^{(63)} \leftarrow \boldsymbol{B}^{(62)}$
$\boldsymbol{D}^{(64)} \leftarrow \boldsymbol{C}^{(63)}$	$D^{(63)} \leftarrow C^{(62)}$
$\boldsymbol{E}^{(64)} \leftarrow D^{(63)} \boxplus T_1^{(64)}$	$\boldsymbol{E}^{(63)} \leftarrow D^{(62)} \boxplus T_1^{(63)}$
$\boldsymbol{F}^{(64)} \leftarrow \boldsymbol{E}^{(63)}$	$\boldsymbol{F}^{(63)} \leftarrow \boldsymbol{E}^{(62)}$
$\boldsymbol{G}^{(64)} \leftarrow \boldsymbol{F}^{(63)}$	$\boldsymbol{G}^{(63)} \leftarrow \boldsymbol{F}^{(62)}$
$\boldsymbol{H}^{(64)} \leftarrow \boldsymbol{G}^{(63)}$	$H^{(63)} \leftarrow G^{(62)}$
Round 62	**Round 61**
$\boldsymbol{A}^{(62)} \leftarrow T_1^{(62)} \boxplus T_2^{(62)}$	$\boldsymbol{A}^{(61)} \leftarrow T_1^{(61)} \boxplus T_2^{(61)}$
$\boldsymbol{B}^{(62)} \leftarrow \boldsymbol{A}^{(61)}$	$B^{(61)} \leftarrow A^{(60)}$
$C^{(62)} \leftarrow B^{(61)}$	$C^{(61)} \leftarrow B^{(60)}$
$D^{(62)} \leftarrow C^{(61)}$	$D^{(61)} \leftarrow C^{(60)}$
$\boldsymbol{E}^{(62)} \leftarrow D^{(61)} \boxplus T_1^{(62)}$	$\boldsymbol{E}^{(61)} \leftarrow D^{(60)} \boxplus T_1^{(61)}$
$\boldsymbol{F}^{(62)} \leftarrow \boldsymbol{E}^{(61)}$	$F^{(61)} \leftarrow E^{(60)}$
$G^{(62)} \leftarrow F^{(61)}$	$G^{(61)} \leftarrow F^{(60)}$
$H^{(62)} \leftarrow G^{(61)}$	$H^{(61)} \leftarrow G^{(60)}$

5.3 Performance Overhead Evaluation

First, the two calls to the compression function dedicated to k_0 and k_1 computations need no security against DPA, so they can be omitted. Then, following the results exposed above, preventing the attack presented in this paper requires countermeasures to protect the intermediate variables of at least the first 20 rounds of each call to the compression function in the inner hash, and of the last 4 rounds of the final call to the compression function in the outer hash. In a first approximation, we leave the details of the implementation for a secure round and simply consider it is k times slower than a non-secure round. In that case, the execution time of an implementation where sensitive rounds of the compression function are protected is approximately $(20k+44)/64 \approx 0,31k$ times slower than an unprotected implementation. Additional work is required to precisely evaluate k, however we expect it to be relatively large. Indeed, if masking is chosen as a countermeasure, switching from arithmetic to boolean masks and backwards (wich is required when arithmetic and boolean operations are mixed, as it is the case for all SHA-1/SHA-2 functions) is usually costly [Mangard et al., 2007].

6 CONCLUSION

We have presented in this paper a side channel attack on HMAC SHA-256 in the Hamming weight model, which requires no assumption on the implementation. Furthermore, it has been seen that this attack can be easily adapted to the Hamming distance model, with less assumptions on the implementation than previous existing attacks. To ensure its feasibility and measure its efficiency, we have simulated the attack. The technique of partial DPA has been used for different operations, and we have estimated the cost of the complete attack depending on the efficiency of the partial attacks. Then, we have analysed the attacks and corresponding protections, and evaluated the performance overhead for software implementations. Further work has to be done to focus on the details of the countermeasures.

Acknowledgments. The authors wish to thank Christophe Giraud for helpful discussions, and anonymous referees of a previous version of this work for their valuable comments.

REFERENCES

Arkko, J. and Haverinen, H. (2006). RFC 4187: Extensible Authentication Protocol Method for 3rd Generation Authentication and Key Agreement (EAP-AKA).

Bellare, M., Canetti, R., and Krawczyk, H. (1996). Keying Hash Functions for Message Authentication. In Koblitz, N., editor, *Advances in Cryptology – CRYPTO '96*, volume 1109 of *LNCS*, pages 1–15. Springer.

Bertoni, G., Daemen, J., Debande, N., Le, T.-H., Peeters, M., and Van Assche, G. (2013). Power Analysis of Hardware Implementations Protected with Secret Sharing. IACR Cryptology ePrint Archive Report 2013/67.

Brassard, G., editor (1989). *Advances in Cryptology – CRYPTO '89*, volume 435 of *LNCS*. Springer.

Brier, E., Clavier, C., and Olivier, F. (2004). Correlation Power Analysis with a Leakage Model. In [Joye and Quisquater, 2004], pages 16–29.

Chari, S., Rao, J., and Rohatgi, P. (2002). Template Attacks. In Kaliski Jr., B., Koç, Ç., and Paar, C., editors, *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of *LNCS*, pages 13–29. Springer.

Clavier, C. and Gaj, K., editors (2009). *Cryptographic Hardware and Embedded Systems – CHES 2009*, volume 5747 of *LNCS*. Springer.

Damgård, I. (1989). A Design Principle for Hash Functions. In [Brassard, 1989], pages 416–427.

FIPS 198-1 (2008). *The Keyed-Hash Message Authentication Code (HMAC)*. National Institute of Standards and Technology.

Fouque, P.-A., Leurent, G., Réal, D., and Valette, F. (2009). Pratical Electromgnetic Template Attack on HMAC. In [Clavier and Gaj, 2009], pages 66–80.

Gauravaram, P. and Okeya, K. (2007). An Update on the Side Channel Cryptanalysis of MACs Based on Cryptographic Hash Functions. In Srinathan, K., Rangan, C. P., and Yung, M., editors, *Progress in Cryptology – INDOCRYPT 2007*, volume 4859 of *LNCS*, pages 393–403. SV.

Gauravaram, P. and Okeya, K. (2008). Side Channel Analysis of Some Hash Based MACs: A Response to SHA-3 Requirements. In Chen, L., Ryan, M. D., and Wang, G., editors, *Information and Communications Security – ICISC 2008*, volume 5308 of *LNCS*, pages 111–127. Springer.

Haverinen, H. and Salowey, J. (2006). RFC 4186: Extensible Authentication Protocol Method for Global System for Mobile Communications (GSM) Subscriber Identity Modules (EAP-SIM).

Joye, M. and Quisquater, J.-J., editors (2004). *Cryptographic Hardware and Embedded Systems – CHES 2004*, volume 3156 of *LNCS*. Springer.

Kocher, P., Jaffe, J., and Jun, B. (1999). Differential Power Analysis. In Wiener, M., editor, *Advances in Cryptology – CRYPTO '99*, volume 1666 of *LNCS*, pages 388–397. Springer.

Lemke, K., Schramm, K., and Paar, C. (2004). DPA on *n*-Bit sized Boolean and Arithmetic Operations and its Application to IDEA, RC6, and the HMAC-Construction. In [Joye and Quisquater, 2004], pages 205–219.

Mangard, S., Oswald, E., and Popp, T. (2007). *Power Analysis Attacks – Revealing the Secrets of Smartcards*. Springer.

McEvoy, R., Tunstall, M., Murphy, C. C., and Marnane, W. P. (2008). Differential Power Analysis of HMAC based on SHA-2, and Countermeasures. In Kim, S., Yung, M., and Lee, H.-W., editors, *WISA 2007*, volume 4867 of *LNCS*, pages 317–332. Springer.

Merkle, R. C. (1989). A Certified Digital Signature. In [Brassard, 1989], pages 218–238.

Messerges, T. (2000). Using Second-order Power Analysis to Attack DPA Resistant Software. In Koç, Ç. and Paar, C., editors, *Cryptographic Hardware and Embedded Systems – CHES 2000*, volume 1965 of *LNCS*, pages 238–251. Springer.

Okeya, K. (2006). Side Channel Attacks Against HMACs Based on Block-Cipher Based Hash Functions. In Batten, L. M. and Safavi-Naini, R., editors, *ACISP*, volume 4058 of *LNCS*, pages 432–443. Springer.

Rivain, M., Prouff, E., and Doget, J. (2009). Higher-Order Masking and Shuffling for Software Implementations of Block Ciphers. In [Clavier and Gaj, 2009], pages 171–188.

Tunstall, M., Hanley, N., McEvoy, R., Whelan, C., Murphy, C., and Marnane, W. (2007). Correlation Power Analysis of Large Word Sizes. In *IET Irish Signals and System Conference – ISSC 2007*, pages 145–150.

Zhang, F. and Shi, Z. J. (2011). Differential and Correlation Power Analysis Attacks on HMAC-Whirlpool. In *ITNG'11*, pages 359–365. IEEE Computer Society.

Zohner, M., Kasper, M., Stöttinger, M., and Huss, S. A. (2012). Side Channel Analysis of the SHA-3 Finalists. In Rosenstiel, W. and Thiele, L., editors, *Design, Automation & Test in Europe Conference & Exhibition, DATE 2012*, pages 1012–1017. IEEE Computer Society.