Differential Power Analysis of HMAC SHA-1 and HMAC SHA-2 in the Hamming Weight Model

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

[1] Oberthur Technologies, 420 rue d'Estienne d'Orves, 92700 Colombes, France
{l.bettale,e.dottax,f.rondepierre}@oberthur.com,
laurie.genelle.p@gmail.com
[2] École Normale Supérieure, 45 rue d'Ulm, 75005 Paris, France
[3] Thales Communications and Security, 4 Avenue des Louvresses,
92230 Gennevilliers, France
sonia.belaid@ens.fr

Abstract. As any algorithm manipulating secret data, HMAC is potentially vulnerable to side channel attacks. In 2004, Lemke et al. fully described a differential power attack on HMAC with RIPEMD-160 in the Hamming weight leakage model, and mentioned a possible extension to SHA-1. Later in 2007, McEvoy et al. proposed an attack against HMAC with hash functions from the SHA-2 family, that works in the Hamming distance leakage model. This attack 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, we give a full description of an extension of this attack to HMAC SHA-1. We also provide a careful study of the protections to develop in order to minimize the impact of the security on the performances.

Keywords: HMAC · Side channel analysis · Differential power analysis · Hamming weight · SHA-1 · SHA-2

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

S. Belaïd and L. Genelle—This work was essentially done while these authors were members of the Cryptography Group of Oberthur Technologies.

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 [3] is the most widely used. Today it is a standardized algorithm [9] and it is used by several protocols running on embedded devices [1,14]. The use of HMAC in such a context leads the research community to study its vulnerability against *Side Channel Analysis* (SCA). 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 [15]. 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 consumption of an embedded device. In the HW model, the leakage is assumed to depend 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 [15,17,20].

Several DPA scenarios have been proposed in the literature to attack the HMAC algorithm. Okeya et al. addressed in several papers [11,12,21] the question of protecting HMAC against DPA. They focused their study on block-cipher based hash functions. As well, [24] dealt with HMAC based on Whirlpool. In [16], Lemke et al. described a theoretical attack on HMAC based on the hash function RIPEMD-160 in the HW model. The authors mentioned that a similar attack on HMAC with SHA-1 is possible following the same approach. McEvoy et al. [18] 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 [10] presented a template attack on HMAC SHA-1, which implies a more powerful adversary than DPA [7]. More recently, SCA on keyed versions of Keccak have been explored in [4,6,23,25].

In this paper, we improve the state of the art on the security of HMAC against DPA by giving a complete description of attacks in the HW model for HMAC with SHA-1 and SHA-2. Contrary to [18], our attacks can be used even when no information about the HMAC implementation is available. We also study the countermeasures required to protect the algorithm, and provide and evaluation of the cost overhead for software implementations.

The rest of the paper is organized as follows. Section 2 introduces the necessary background on HMAC, SHA-2 and SHA-1 algorithms. Section 3 describes the attacks in details. Section 4 deals with the protections required to secure HMAC implementations against our attacks, and notably it evaluates the impact on performances. Finally, Sect. 5 concludes the paper.

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 [9], it is defined as follows:

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

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 $\mathrm{H}\left((k \oplus \mathit{ipad}) \parallel m\right)$ 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 [8,19] (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 $\mathrm{F}\left(\mathrm{IV}, k \oplus \mathit{ipad}\right)$, with IV being the public *Initial Value* of the hash function. The final chaining value $\mathrm{CV_N}$, also denoted by z, is the input of the outer hash. It is processed with a second fixed key-dependent value $k_0 = \mathrm{F}\left(\mathrm{IV}, k \oplus \mathit{opad}\right)$ in the last call of the compression function that outputs the MAC. So we rewrite the HMAC procedure as follows:

$$\mathrm{HMAC}_k\left(m\right) = \mathrm{F}\left(k_0, \mathrm{F}\left(\ldots \mathrm{F}\left(\mathrm{F}\left(k_1, m_1\right), m_2\right), \ldots, m_N\right) \parallel \mathit{pad}\right),$$

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 resume our analysis on the HMAC algorithm based on SHA-256 as presented in [2] and we extend it to HMAC-SHA-1. We assume F to be the SHA-256 or SHA-1 compression function. Brief descriptions of both functions are given in the next section.

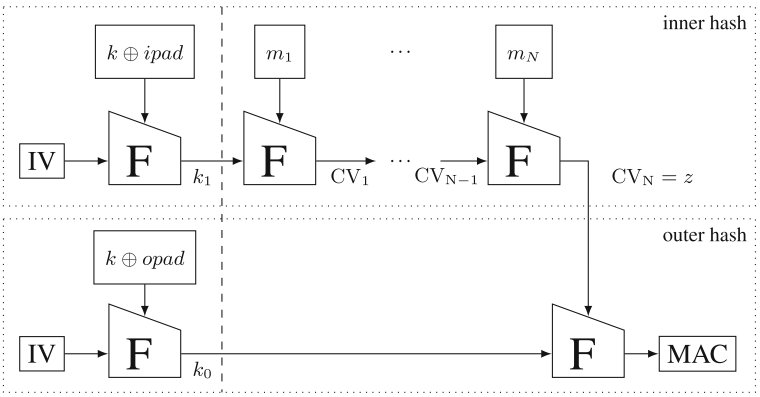

Fig. 1. HMAC using a Merkle-Damgård hash function.

2.2 The SHA-256 Compression Function

As the compression functions of SHA-256 and SHA-512 are exactly the same except for the size of the internal variables (32-bit for SHA-256 against 64-bit for SHA-512), we limit ourselves to the description of the SHA-256 case.

The SHA-256 compression function F is described in Algorithm 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 $V = (V_1, \dots, V_8)$. 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:

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

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 $M_1, \dots, M_{16}$, and expands it into 64 words W_t by using the following additional 32-bit words operations:

$$\begin{aligned}
\sigma_0\,(u) &= (u \ggg 7) \oplus (u \ggg 18) \oplus (u \gg 3) \\
\sigma_1\,(u) &= (u \ggg 17) \oplus (u \ggg 19) \oplus (u \gg 10)
\end{aligned}$$

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

2.3 The SHA-1 Compression Function

The SHA-1 compression function is described in Algorithm 2. It accepts as input a 512-bit message block M and a 160-bit chaining value V (i.e., parameters n and h in Sect. 2.1 equal 512 and 160 respectively). The function iterates 80 times the same round transformation on an internal state. The state is represented by eight 32-bit words A, B, C, D and E initially filled with $V = (V_1, \dots, V_5)$. The round is a composition of 32-bit modular additions, with boolean operations which are defined on 32-bit words u, v and w for each round t as follows:

Algorithm 1. SHA-256 Compression Function.

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

```
 1: $(W_1, \dots, W_{16}) \leftarrow (M_1, \dots, M_{16})$
 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: $(A, B, C, D, E, F, G, H) \leftarrow (V_1, \dots, V_8)$
 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 $(V_1 \boxplus A, \dots, V_8 \boxplus H)$                      ▷ Final Addition
```

$$f_t(u, v, w) = \mathrm{Ch}(u, v, w), \text{ for } 1 \leqslant t \leqslant 20,$$
$$f_t(u, v, w) = \mathrm{Maj}(u, v, w), \text{ for } 41 \leqslant t \leqslant 60,$$
$$f_t(u, v, w) = u \oplus v \oplus w, \text{ for } 21 \leqslant t \leqslant 40, \text{ and } 61 \leqslant t \leqslant 80.$$

3 DPA on HMAC SHA-256 and HMAC SHA-1

3.1 Related Work and Contribution

In [18], 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 Algorithm 1, and in particular they make the following assumptions. Firstly, the variables $A, B, \dots, H$ are initialized with the input chaining value and T_1 is initialized with an unknown but constant value. Secondly, each one of the variables $T_1, T_2, A, B, \dots, H$ 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 Algorithm 1) and rather choose to directly update the pointers, which would clearly be more efficient.

Algorithm 2. SHA-1 compression function.

```
Inputs: the data block $M = (M_1, \ldots, M_{16})$,      the chaining value $V = (V_1, \ldots, V_5)$
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 $80$ do                                   ▷ Message Expansion
 3:     $W_t \leftarrow (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16})_{\lll 1}$
 4: end for
 5: $(A, B, C, D, E) \leftarrow (V_1, \ldots, V_5)$
 6: for $t = 1$ to $80$ do                                    ▷ Main Loop
 7:     $T \leftarrow A_{\lll 5} \boxplus f_t(B, C, D) \boxplus E \boxplus K_t \boxplus W_t$
 8:     $E \leftarrow D$
 9:     $D \leftarrow C$
10:     $C \leftarrow B_{\lll 30}$
11:     $B \leftarrow A$
12:     $A \leftarrow T$
13: end for
14: return $(V_1 \boxplus A,\ V_2 \boxplus B,\ V_3 \boxplus C,\ V_4 \boxplus D,\ V_5 \boxplus E)$   ▷ Final Addition
```

In [16], the authors apply partial DPAs to several algorithm, among which HMAC RIPEMD and HMAC SHA-1. They only describe the steps for RIPEMD, but their attack is dependent on the order in which the operations are performed.

In this paper, we propose an attack on the compression function that targets different steps in the algorithm compared to [18]. 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 [17]. Secondly, we show in Sect. 3.4 how a similar attack can be mounted on HMAC SHA-1. In particular, this attack does not need any assumption on the order of the operations.

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

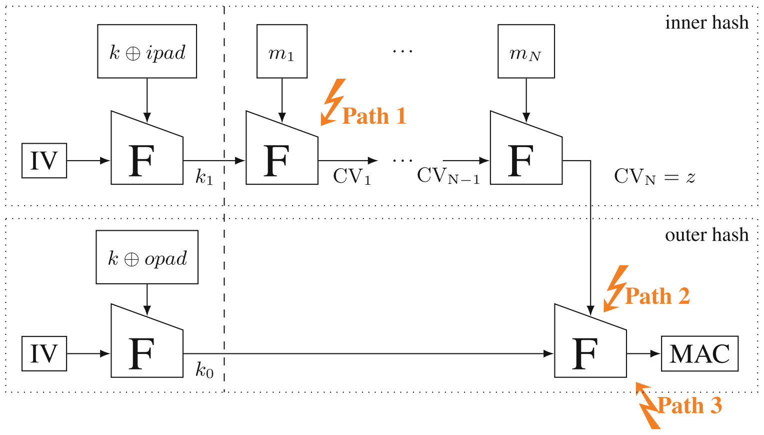

Fig. 2. Attack paths on HMAC.

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, we give the detailed steps of the attacks following respectively Path 1, 2 and 3.

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.*

3.3 Attack on HMAC SHA-2

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 Algorithm 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.

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

Rnd	Attack	Targeted operation	Var.	Hyp.	Result
1	DPA 1	$T_1^{(1)} \leftarrow \delta^{(0)} \boxplus W_1$	$\widehat{W_1}$	$\delta^{(0)}$	$\widehat{T}_1^{(1)}, \delta^{(0)}$
	DPA 2	$E^{(1)} \leftarrow D^{(0)} \boxplus T_1^{(1)}$	$\widehat{T}_1^{(1)}$	$D^{(0)}$	$\widehat{E}^{(1)}$, $\mathbf{D^{(0)}}$
	DPA 3	$A^{(1)} \leftarrow T_1^{(1)} \boxplus T_2^{(1)}$	$\widehat{T}_1^{(1)}$	$T_2^{(1)}$	$\widehat{A}^{(1)}, T_2^{(1)}$
2	DPA 4	$E^{(1)} \wedge F^{(1)}$ in $\mathrm{Ch}\left(E^{(1)}, F^{(1)}, G^{(1)}\right)$	$\widehat{E}^{(1)}$	$F^{(1)}$	$F^{(1)} = \mathbf{E^{(0)}}$
	DPA 5	$\overline{E^{(1)}} \wedge G^{(1)}$ in $\mathrm{Ch}\left(E^{(1)}, F^{(1)}, G^{(1)}\right)$	$\widehat{E}^{(1)}$	$G^{(1)}$	$G^{(1)} = \mathbf{F^{(0)}}$
	DPA 6	$A^{(1)} \wedge B^{(1)}$ in $\mathrm{Maj}\left(A^{(1)}, B^{(1)}, C^{(1)}\right)$	$\widehat{A}^{(1)}$	$B^{(1)}$	$B^{(1)} = \mathbf{A^{(0)}}$
	DPA 7	$A^{(1)} \wedge C^{(1)}$ in $\mathrm{Maj}\left(A^{(1)}, B^{(1)}, C^{(1)}\right)$	$\widehat{A}^{(1)}$	$C^{(1)}$	$C^{(1)} = \mathbf{B^{(0)}}$
	DPA 8	$T_1^{(2)} \leftarrow H^{(1)} \boxplus \Sigma_1\left(E^{(1)}\right) \boxplus \mathrm{Ch}\left(E^{(1)}, F^{(1)}, G^{(1)}\right) \boxplus K_2 \boxplus W_2$	$\widehat{W_2}$	$H^{(1)}$	$H^{(1)} = \mathbf{G^{(0)}}$

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.

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.

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.

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:

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

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 $A^{(64)}$, ..., $H^{(64)}$ 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.

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 [16] as explained in [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.

3.4 Attack on HMAC SHA-1

The attack can be adapted for SHA-1. In particular, the attack related to Path 3 can be straightforwardly used for SHA-1 as it also implements a final addition between the secret k_0 and the known output.

The attack related to Path 1 and 2 needs more adaptation. We develop the attack in Table 2 with the same notations as Table 1. We also use the notations $\beta^{(i)}$ and $\gamma^{(i)}$ to ease the notations of the sum $A^{(i)}_{\lll 5} \boxplus \mathrm{Ch}\left(B^{(i)}, C^{(i)}, D^{(i)}\right) \boxplus E^{(i)} \boxplus K_{i+1} = A^{(i)} \boxplus \beta^{(i)} = \gamma^{(i)}$. Using these DPAs, the attacker recovers the following parts of the secret: $A^{(0)}, B^{(0)}, C^{(0)}$. The last remaining parts $D^{(0)}$ and $E^{(0)}$ can be recovered by making substitutions in Algorithm 2: in Step 7 of round 2, where $E^{(1)} = D^{(0)}$ is the only unknown variable, and similarly in Step 7 of round 1 where $E^{(0)}$ becomes the only unknown variable.

Full Attack. Only 5 DPAs are needed to recover a full state of SHA-1. Following the same paths as in SHA-2, this amounts to 10 DPAs on 32-bit words.

Table 2. DPA attack on SHA-1 compression function using HW leakage model.

Rnd	Attack	Targeted operation	Var.	Hyp.	Result
1	DPA 1	$T^{(1)} \leftarrow A^{(0)}_{\lll 5} \boxplus \mathrm{Ch}\left(B^{(0)}, C^{(0)}, D^{(0)}\right) \boxplus E^{(0)} \boxplus K_1 \boxplus W_1$	$\widehat{W}_1$	$\gamma^{(0)}$	$\widehat{T}^{(1)} = \widehat{A}^{(1)} = \widehat{B}^{(2)}$
2	DPA 2	$T^{(2)} \leftarrow A^{(1)}_{\lll 5} \boxplus \mathrm{Ch}\left(B^{(1)}, C^{(1)}, D^{(1)}\right) \boxplus E^{(1)} \boxplus K_2 \boxplus W_2$	$\widehat{W}_2$	$\beta^{(1)}$	$\widehat{T}^{(2)} = \widehat{A}^{(2)}$
3	DPA 3	$B^{(2)} \wedge C^{(2)}$ in $\mathrm{Ch}\left(B^{(2)}, C^{(2)}, D^{(2)}\right)$	$\widehat{B}^{(2)}$	$C^{(2)}$	$C^{(2)} = \mathbf{A}^{(\mathbf{0})}$
	DPA 4	$\overline{B^{(2)}} \wedge D^{(2)}$ in $\mathrm{Ch}\left(B^{(2)}, C^{(2)}, D^{(2)}\right)$	$\widehat{B}^{(2)}$	$D^{(2)}$	$D^{(2)} = \mathbf{B}^{(\mathbf{0})}$
	DPA 5	$T^{(3)} \leftarrow A^{(2)}_{\lll 5} \boxplus \mathrm{Ch}\left(B^{(2)}, C^{(2)}, D^{(2)}\right) \boxplus E^{(2)} \boxplus K_3 \boxplus W_3$	$\widehat{W}_3$	$B^{(1)}$	$E^{(2)} = \mathbf{C}^{(\mathbf{0})}$

3.5 Summary

Contrarily to previously known attacks on HMAC, the attacks described in this section are sound in the Hamming weight model without any assumption on the implementation. Furthermore, our attack can be straightforwardly adapted to the Hamming distance model (see [2]). Simulations and complexity study of our attack on SHA-2 using partial DPA can also be found in [2].

4 Protected Implementation

In this section, we focus on the countermeasures to apply to secure a HMAC implementation instantiated with SHA-1, or a function from the SHA-2 family, against the described attacks (and the ones in [16,18]). In software, the main techniques used to thwart such SCA are *masking* and *shuffling*, as well as combination of both [22]. 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 [18], the authors use the hardware-specific techniques from [13] for masking and propose a secure implementation of HMAC SHA-256 on FPGA. It is based on a completely masked SHA-256 algorithm, where all intermediate values are randomized. In particular, the authors did not investigate whether some calls to the compression function or some rounds inside the compression function could be left unmasked. This approach is acceptable in hardware. Indeed, the implementation of both masked and unmasked versions of some functions would have a substantial impact on the area used by the circuit. Given that hardware resources must usually be minimized for cost reasons, and that the performance overhead induced by masking is not excessive in hardware, securing the whole algorithm constitutes an acceptable trade-off.

This is in opposition to software implementations. In this case, code is not so much a scarce resource, while masking has an important impact on performances. It is thus worth carefully studying the algorithm and establishing whether some parts can be left unmasked. We show hereafter that the HMAC computation does not need complete protection and that an efficient secure software implementation can be achieved.

In the rest of this section, we show how to prevent any information leakage on the secret values k_0 and k_1 with a minimal impact on performances. We then provide an evaluation of the timing overhead induced by the countermeasures.

4.1 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 and to mount a DPA on the outer hash compression function. Protecting only the latter, we leave the secret key k_1 to the attacker, which is not satisfactory. Therefore we choose to protect the generation of values z. 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. Since it is specific to the compression function, we consider separately SHA-2 family and SHA-1 hash functions.

SHA-2. We focus here on SHA-256 since SHA-512 behaves exactly the same with a twice larger block size and an greater number of rounds and SHA-224 and SHA-384 are essentially truncated versions of the same function. The attacks following Path 1 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. But, as we can see in Algorithm 1, parts of the input message block are also involved in each of the other 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 values $E^{(t-1)}, F^{(t-1)}, G^{(t-1)}$, $H^{(t-1)}$ 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 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 and the value W_t must be variable. To achieve the first condition, variables $(W_1, \ldots, W_{t-1})$ associated to the previous rounds 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 fulfilled only for $t \leqslant 16$. We conclude that the attack from Path 1 presented in Sect. 3.3 can be extended to any rounds among the first 16 ones. 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.

SHA-1. We now focus on the slightly different hash function SHA-1. As depicted in Sect. 3.2, the attacks following Paths 1 and 2 require this time the observation of rounds 1 to 3 of the compression function. We thus need to protect the sensitive variables manipulated in these three rounds. Furthermore, since the input message block is involved in each of the 80 rounds, we have to check the feasibility of this attack on each of them. We assume that the attacker applies the attack described in Table 2 to rounds t to $t+2$, with $t \geqslant 2$. The first attack DPA 1 works the same and gives the attacker the control on $A^{(t)}$ when the intermediate values $A^{(t-1)}$, $B^{(t-1)}$, $C^{(t-1)}$, $D^{(t-1)}$ and $E^{(t-1)}$ are fixed and block W_t varies. Under the same assumptions, an adaptation of DPA 2 can be performed and gives the attacker the control on $A^{(t+1)}$. DPAs 3, 4 and 5 can also be adapted and substitutions can be made to recover the whole internal state. Afterwards, the attacker just has to invert the round function to recover the secret input chaining value V.

Thus, the tweaked attack works if values $A^{(t-1)}$, $B^{(t-1)}$, $C^{(t-1)}$, $D^{(t-1)}$, $E^{(t-1)}$ are fixed and the value W_t is variable. Since these two requirements can only be fulfilled for $t \leqslant 16$, the attack can be extended to any rounds among the 16 first ones. But as some of the sensitive variables produced at round 16 remain available until the 21th round, the protection to thwart the attack must be applied to rounds 1 to 21.

4.2 Preventing Path 3

Section 3.3 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 the sensitive variables for the two families of hash functions.

SHA-2 and SHA-1. The sensitive variables targeted by the attack are:

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

for SHA-2 family and

$$A^{(\tau)} = R_1 \boxminus V_1, \quad B^{(\tau)} = R_2 \boxminus V_2, \quad \ldots, \quad E^{(\tau)} = R_5 \boxminus V_5$$

for SHA-1 family where τ denotes the number of rounds (64 for SHA-256 and 80 for SHA-512 and SHA-1), 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 3 for SHA-2 and in Table 4 for SHA-1. This shows that sensitive variables in SHA-2 are produced in round $(\tau - 3)$, thus protection is required in the last four rounds. For SHA-1 family hash functions, the sensitive variables are produced in round $(\tau - 4)$ thus protection is required in the last five rounds.

Table 3. Sensitive variables in last rounds of SHA-2.

Roundτ	$\boldsymbol{A^{(\tau)}}$	$\boldsymbol{B^{(\tau)}}$	$\boldsymbol{C^{(\tau)}}$	$\boldsymbol{D^{(\tau)}}$	$\boldsymbol{E^{(\tau)}}$	$\boldsymbol{F^{(\tau)}}$	$\boldsymbol{G^{(\tau)}}$	$\boldsymbol{H^{(\tau)}}$
Round$(\tau - 1)$	$\boldsymbol{A^{(\tau-1)}}$	$\boldsymbol{B^{(\tau-1)}}$	$\boldsymbol{C^{(\tau-1)}}$	$D^{(\tau-1)}$	$\boldsymbol{E^{(\tau-1)}}$	$\boldsymbol{F^{(\tau-1)}}$	$\boldsymbol{G^{(\tau-1)}}$	$H^{(\tau-1)}$
Round$(\tau - 2)$	$\boldsymbol{A^{(\tau-2)}}$	$\boldsymbol{B^{(t-2)}}$	$C^{(\tau-2)}$	$D^{(\tau-2)}$	$\boldsymbol{E^{(\tau-2)}}$	$\boldsymbol{F^{(\tau-2)}}$	$G^{(\tau-2)}$	$H^{(\tau-2)}$
Round$(\tau - 3)$	$\boldsymbol{A^{(\tau-3)}}$	$B^{(\tau-3)}$	$C^{(\tau-3)}$	$D^{(\tau-3)}$	$\boldsymbol{E^{(\tau-3)}}$	$F^{(\tau-3)}$	$G^{(\tau-3)}$	$H^{(\tau-3)}$

4.3 Considering Other Paths

The approach presented above is secure only if no attack path exists that could be used to mount an attack on unprotected rounds. We first focus on the unprotected part of the compression function SHA-2, and then examine the case of SHA-1.

Table 4. Sensitive variables in last rounds of SHA-1.

Roundτ	$\boldsymbol{A^{(t)}}$	$\boldsymbol{B^{(t)}}$	$\boldsymbol{C^{(t)}}$	$\boldsymbol{D^{(\tau)}}$	$\boldsymbol{E^{(\tau)}}$
Round$(\tau-1)$	$\boldsymbol{A^{(\tau-1)}}$	$\boldsymbol{B^{(\tau-1)}}$	$\boldsymbol{C^{(\tau-1)}}$	$\boldsymbol{D^{(\tau-1)}}$	$E^{(\tau-1)}$
Round$(\tau-2)$	$\boldsymbol{A^{(\tau-2)}}$	$\boldsymbol{B^{(\tau-2)}}$	$\boldsymbol{C^{(\tau-2)}}$	$D^{(\tau-2)}$	$E^{(\tau-2)}$
Round$(\tau-3)$	$\boldsymbol{A^{(\tau-3)}}$	$\boldsymbol{B^{(\tau-3)}}$	$C^{(\tau-3)}$	$D^{(\tau-3)}$	$E^{(\tau-3)}$
Round$(\tau-4)$	$\boldsymbol{A^{(\tau-4)}}$	$B^{(\tau-4)}$	$C^{(\tau-4)}$	$D^{(\tau-4)}$	$E^{(\tau-4)}$

SHA-2. Let us have a more general look at the algorithm. To mount a DPA, it is necessary to target an operation where a known variable is mixed with a secret, and to be able to predict the result of this operation according to hypotheses on the value of the secret. The only known variables that we can vary are the W_t, which are introduced during the computation of T_1 and then propagated in the internal state. Section 4.1 has considered natural extensions of DPA 1, where the first $t-1$ blocks of the message are fixed while the t-th one is used as the variable. However, it has been shown that for $t > 16$, no W_t can vary without changing one of the previous blocks. If we consider an implementation secure regarding Path 1, an attacker has to target an operation after round 20. She thus has no other possibility than varying a block W_t and targeting an operation *after* the corresponding $T_1^{(t)}$ has been computed. In particular, she will have to express internal results as functions of W_t and some secret data, despite the mixing that will already have occurred. Let us consider the simplest case: blocks W_1 to W_{15} and W_{17} are fixed, while block 16 varies. At round 21, the following value is manipulated:

$$H^{(20)} = D^{(16)} \boxplus H^{(16)} \boxplus \Sigma_1\left(E^{(16)}\right) \boxplus \mathrm{Ch}\left(E^{(16)}, F^{(16)}, G^{(16)}\right) \boxplus K_{17} \boxplus W_{17}.$$

In this equation, variables $D^{(16)}, F^{(16)}, G^{(16)}$ and $H^{(16)}$ are fixed, while $E^{(16)}$ depends on W_{16} and on a sum of fixed values that we refer to as Δ. As it is computationally impossible to make hypotheses directly on all involved secret values, $D^{(16)} + H^{(16)}, F^{(16)}, G^{(16)}$ and Δ[1], we search for simpler relations and consider mounting a partial DPA. Assuming we can perform a partial DPA on one bit, we focus on the least significant bit (LSB) of $H^{(20)}$. It depends, among others, on the LSB b of $\Sigma_1(\Delta \boxplus W_{16})$. To simplify we write W for W_{16}, and we note $X_{(i)}$ the i-th bit of X. Then we have:

$$b = \Delta_{(6)} \oplus W_{(6)} \oplus c_6 \oplus \Delta_{(11)} \oplus W_{(11)} \oplus c_{11} \oplus \Delta_{(25)} \oplus W_{(25)} \oplus c_{25},$$

where c_i denotes the carry that propagates during the addition $\Delta \boxplus W$. As these carries depend on W, we have to make additional assumptions on all involved bits of Δ (expressing the carries as functions of bits of Δ and W does not lead to a reduced number of hypothesis). For this sole bit b, we already have to make 26

[1] We cannot target the intermediate computations since they are performed in secure rounds.

bits of hypothesis. Coming back to $H^{(20)}$, we have to make a 29-bit hypothesis to recover only the LSB. Note that considering other variables $(A^{(20)}, B^{(20)}, \dots)$ or different rounds does not help. In all cases, at least the same hypotheses have to be made since the variable W is introduced via the variable T_1.

Such an attack based on this partial DPA would be very complex to mount in practice. We thus reasonably assume that no additional protection is needed and a safe and efficient implementation can be achieved.

SHA-1. In the case of SHA-1, the first 21 rounds and the last 5 ones must be protected. Therefore, if we assume the implementation secure against the given attacks, the attacker can only target an operation between rounds 22 and 75. Let us consider here the simplest case when W_1 to W_{15} and W_{17} are fixed and W_{16} varies. At round 22, the following value is manipulated:

$$E^{(21)} = A^{(17)} = A^{(16)}_{\lll 5} \boxplus F(B^{(16)}, C^{(16)}, D^{(16)}) \boxplus K_{17} \boxplus W_{17}.$$

In this equation, all the variables are fixed but $A^{(16)}$ which depends on W_{16}. This time again, we cannot make hypotheses on all the other secret values. Furthermore, regarding only the LSB of $E^{(21)}$, we already need to make a 29-bit hypothesis. As for the SHA-2 family, considering other variables does not help. Indeed at least the same hypotheses have to be made since the message block is always introduced via the variable T. For the same reasons as for SHA-2 family hash functions, the complexity of the simplest attack justifies the absence of additional protection to achieve a secure implementation.

4.4 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 is possible while leaving completely unmasked the main part of each treated instantiation of HMAC. Equivalently, protecting an implementation only requires countermeasures on

- the first t_0 rounds of each call to the compression function in the inner hash (where t_0 equals 20 for SHA-2 and 21 for SHA-1);
- the last t_1 rounds of the final call to the compression function in the outer hash (where t_1 equals 4 for SHA-2 and 5 for SHA-1).

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 $(t_0\,k+(\tau-t_0))/\tau \approx 0,31\,k$ times slower than an unprotected implementation, where t is the number of rounds. 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 (which is required when arithmetic and boolean operations are mixed, as it is the case for all SHA-1/SHA-2 functions) is usually costly [17].

5 Conclusions

We have presented in this paper side channel attacks on HMAC with SHA-1 and SHA-2 in the Hamming weight model. The complete attack steps have been described. Then, we have analysed the attacks and corresponding protections, and proposed a strategy that limits the performance overhead for software implementations. Better than masking everything, we have determined which parts of the HMAC algorithm actually need protection. Further work has to be done to focus on the details of the countermeasures.

Acknowledgements. 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

1. Arkko, J., Haverinen, H.: RFC 4187: Extensible Authentication Protocol Method for 3rd Generation Authentication and Key Agreement (EAP-AKA) (2006)
2. Belaïd, S., Bettale, L., Dottax, E., Genelle, L., Rondepierre, F.: Differential power analysis of HMAC SHA-2 in the Hamming weight model. In: Samarati, P. (ed.) SECRYPT, SECRYPT is Part of ICETE - The International Joint Conference on e-Business and Telecommunications, pp. 230–241. SciTePress, USA (2013)
3. Bellare, M., Canetti, R., Krawczyk, H.: Keying hash functions for message authentication. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 1–15. Springer, Heidelberg (1996)
4. Bertoni, G., Daemen, J., Debande, N., Le, T.H., Peeters, M., Van Assche, G.: Power Analysis of Hardware Implementations Protected with Secret Sharing. IACR Cryptology ePrint Archive Report 2013/67 (2013). http://eprint.iacr.org/2013/67. A preliminary version appeared at MICROW'12 [5]
5. Bertoni, G., Daemen, J., Debande, N., Le, T. H., Peeters, M., Van Assche, G.: Power analysis of hardware implementations protected with secret sharing. In: 45th Annual IEEE/ACM International Symposium on Microarchitecture Workshops (MICROW), pp. 9–16. IEEE Computer Society (2012)
6. Bettale, L., Dottax, E., Genelle, L., Piret, G.: Collision-correlation attack against a first-order masking scheme for MAC based on SHA-3. In: Prouff, E. (ed.) COSADE 2014. LNCS, vol. 8622, pp. 129–143. Springer, Heidelberg (2014)
7. Chari, S., Rao, J., Rohatgi, P.: Template attacks. In: Kaliski Jr., B., Koç, Ç., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2002. LNCS, vol. 2523, pp. 13–29. Springer, Heidelberg (2002)
8. Damgård, I.B.: A design principle for hash functions. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 416–427. Springer, Heidelberg (1990)
9. FIPS 198–1: The Keyed-Hash Message Authentication Code (HMAC). National Institute of Standards and Technology, July 2008

10. Fouque, P.-A., Leurent, G., Réal, D., Valette, F.: Practical electromagnetic template attack on HMAC. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 66–80. Springer, Heidelberg (2009)
11. Gauravaram, P., Okeya, K.: An update on the side channel cryptanalysis of MACs based on cryptographic hash functions. In: Srinathan, K., Rangan, C.P., Yung, M. (eds.) INDOCRYPT 2007. LNCS, vol. 4859, pp. 393–403. Springer, Heidelberg (2007)
12. Gauravaram, P., Okeya, K.: Side channel analysis of some hash based MACs: a response to SHA-3 requirements. In: Chen, L., Ryan, M.D., Wang, G. (eds.) ICICS 2008. LNCS, vol. 5308, pp. 111–127. Springer, Heidelberg (2008)
13. Golić, J.D.: Techniques for random masking in hardware. IEEE Trans. Circ. Syst. I **54**(2), 291–300 (2007)
14. Haverinen, H., Salowey, J.: RFC 4186: Extensible Authentication Protocol Method for Global System for Mobile Communications (GSM) Subscriber Identity Modules (EAP-SIM) (2006)
15. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)
16. Lemke, K., Schramm, K., Paar, C.: DPA on n-bit sized Boolean and arithmetic operations and its application to IDEA, RC6, and the HMAC-Construction. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 205–219. Springer, Heidelberg (2004)
17. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks - Revealing the Secrets of Smartcards. Springer, US (2007)
18. McEvoy, R., Tunstall, M., Murphy, C.C., Marnane, W.P.: Differential power analysis of HMAC based on SHA-2, and countermeasures. In: Kim, S., Yung, M., Lee, H.-W. (eds.) WISA 2007. LNCS, vol. 4867, pp. 317–332. Springer, Heidelberg (2008)
19. Merkle, R.C.: A certified digital signature. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 218–238. Springer, Heidelberg (1990)
20. Messerges, T.S.: Using second-order power analysis to attack DPA resistant software. In: Paar, C., Koç, Ç.K. (eds.) CHES 2000. LNCS, vol. 1965, pp. 238–251. Springer, Heidelberg (2000)
21. Okeya, K.: Side channel attacks against HMACs based on block-cipher based hash functions. In: Batten, L.M., Safavi-Naini, R. (eds.) ACISP 2006. LNCS, vol. 4058, pp. 432–443. Springer, Heidelberg (2006)
22. Rivain, M., Prouff, E., Doget, J.: Higher-order masking and shuffling for software implementations of block ciphers. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 171–188. Springer, Heidelberg (2009)
23. Taha, M., Schaumont, P.: Side-channel analysis of MAC-Keccak. In: IEEE International Symposium on Hardware-Oriented Security and Trust - HOST 2013. IEEE Computer Society (2013)
24. Zhang, F., Shi, Z. J.: Differential and correlation power analysis attacks on HMAC-Whirlpool. In: ITNG 2011, pp. 359–365. IEEE Computer Society (2011)
25. Zohner, M., Kasper, M., Stöttinger, M., Huss, S.A.: Side channel analysis of the SHA-3 finalists. In: Rosenstiel, W., Thiele, L. (eds.) Design, Automation & Test in Europe Conference & Exhibition, DATE 2012, pp. 1012–1017. IEEE Computer Society, USA (2012)