Practical Electromagnetic Template Attack on HMAC

Pierre-Alain Fouque[1], Gaëtan Leurent[1], Denis Réal[2,3], and Frédéric Valette[3]

[1] École normale supérieure/CNRS/INRIA
{Pierre-Alain.Fouque,Gaetan.Leurent}@ens.fr
[2] INSA-IETR, 20 avenue des coesmes, 35043 Rennes, France
Denis.Real@insa-rennes.fr
[3] CELAR, 35 Bruz, France
{Denis.Real,Frederic.Valette}@dga.defense.gouv.fr

Abstract. In this paper, we show a very efficient side channel attack against HMAC. Our attack assumes the presence of a side channel that reveals the Hamming distance of some registers. After a *profiling phase* in which the adversary has access to a device and can configure it, the attack recovers the secret key by monitoring a *single execution* of HMAC-SHA-1. The secret key can be recovered using a "template attack" with a computation of about $2^{32}3^{\kappa}$ compression functions, where κ is the number of 32-bit words of the key. Finally, we show that our attack can also be used to break the secrecy of network protocols usually implemented on embedded devices.

We have performed experiments using a NIOS processor executed on a Field Programmable Gate Array (FPGA) to confirm the leakage model. We hope that our results shed some light on the requirements in term of side channel attack for the future SHA-3 function.

1 Introduction

HMAC is a hash-based message authentication code proposed by Bellare, Canetti and Krawczyk in 1996 [3]. It is very interesting to study for at least three reasons: HMAC is standardized (by ANSI, IETF, ISO and NIST) and widely deployed (*e.g.* SSL, TLS, SSH, IPsec); HMAC has security proofs [2,3]; and it is a rather simple construction. It is used in a lot of Internet standards. For instance embedded devices running IPsec protocols [16] have to implement it. There are many such efficient equipments on the market from router vendors that incorporate security protocols on their systems. It is crucial to study the security of such implementations since Virtual Private Network (VPN) products are widely deployed and used to secure important networks.

Recently, new attacks on HMAC based on Wang *et al.* [31,29,30,28] collision attacks have emerged. However, either their complexity is very high, or they attack a function no more widely used in practice such as HMAC-MD4, or the security model is not really practical such as the related key model [6,24,8]. Here, we focus on more practical attacks on HMAC. We show that when HMAC is implemented on embedded devices, and the attacker has access to the physical device to use a side channel, then there exist devastating attacks. These kind of attacks do not rely on previous collision attacks and can be applied on many hash functions, such as MD5 or SHA-2 for instance. Even though side channel attacks are believed to be very intrusive techniques, we show that our attack can be mounted using little contact with the targeted device.

We choose to illustrate our attack with HMAC since it is used in a lot of Internet standards. Beyond the integrity attacks, the side channel attack we describe, can be used to attack the confidentiality of other Internet standards such as the Layer Two Tunneling Protocol (L2TP [27]) or to attack the key derivation of IPsec in the Internet Key Exchange (IKE [13]) protocol. Our attack can also be applied on a famous side channel countermeasure, proposed by Kocher in [18] which is to derive a specific key for each call to the cryptographic application. This kind of protection is very efficient against classical DPA techniques and makes previous attacks infeasible. However our attack allows to recover the master key after listening to only two derivation processes.

1.1 Description of SHA-1

All the computations in SHA-1 are made on 32-bit words. We use $\boxplus$ to denote the modular addition, and $X^{\lll n}$ to denote the bit-wise rotation of X by n bits. SHA-1 is an iterated hash function following the Merkle-Damgård paradigm. The message is padded and cut into blocks of k bits (with $k = 512$ for SHA-1), and the digest is computed by iterating a compression function, starting with an initial value IV.

The compression function of SHA-1 is an unbalanced Feistel ladder with an internal state of five 32-bit registers A, B, C, D, E. The compression function has two inputs: a chaining value which is used as the initial value of the internal registers A_{-1}, B_{-1}, C_{-1}, D_{-1}; and a message block cut into 16 message words $M_0 ... M_{15}$. The message is expanded into 80 words $W_0 ... W_{79}$, such that $W_i = M_i$ for $i < 16$. Then we iterate 80 steps, where each step updates one of the registers. Each step uses one word W_i of the expanded message. If we use A_i, B_i, C_i, D_i, E_i to denote the value of the registers after the step i, the compression function of SHA-1 can be described by:

Step update:	$A_{i+1} = \Phi_i \boxplus A_i^{\lll 5} \boxplus W_i \boxplus E_i \boxplus K_i$				
:	$\Phi_i = f_i(B_i, C_i, D_i)$				
:		$B_{i+1} = A_i$	$C_{i+1} = B_i^{\lll 30}$	$D_{i+1} = C_i$	$E_{i+1} = D_i$
Input:	$A_{-1} \parallel$	$B_{-1} \parallel$	$C_{-1} \parallel$	$D_{-1} \parallel$	E_{-1}
Output:	$A_{-1} \boxplus A_{79} \parallel$	$B_{-1} \boxplus B_{79} \parallel$	$C_{-1} \boxplus C_{79} \parallel$	$D_{-1} \boxplus D_{79} \parallel$	$E_{-1} \boxplus E_{79}$

1.2 Description of HMAC

HMAC is a hash-based message authentication code proposed by Bellare, Canetti and Krawczyk [3]. Let H be an iterated Merkle-Damgård hash function. HMAC is defined by

$$\mathrm{HMAC}_k(M) = H(\bar{k} \oplus \mathsf{opad} \,||\, H(\bar{k} \oplus \mathsf{ipad} \,||\, M)),$$

where M is the message, k is the secret key, $\bar{k}$ its completion to a single block of the hash function, ipad and opad are two fixed one-block values.

The security of HMAC is based on that of NMAC. Since H is assumed to be based on the Merkle-Damgård paradigm, we denote by H_k the modification of H where the public IV is replaced by the secret key k. Then NMAC with secret key (k_1, k_2) is defined by:

$$\mathrm{NMAC}_{k_1,k_2}(M) = H_{k_1}(H_{k_2}(M)).$$

We call k_2 the inner key, and k_1 the outer key. Due to the iterative structure of H, HMAC_k is essentially equivalent to $\mathrm{NMAC}_{H(\bar{k}\oplus\mathsf{opad}),H(\bar{k}\oplus\mathsf{ipad})}$.

Any key-recovery attack against NMAC can be used to recover an equivalent inner key $H(\bar{k} \oplus \mathsf{ipad})$ and an equivalent outer key $H(\bar{k} \oplus \mathsf{opad})$ in HMAC. This information is equivalent to the key of HMAC, since it is sufficient to compute any MAC. Most previous attacks against HMAC are of this kind, but our attack is different: we will use a side channel during the computation of $H(\bar{k} \oplus \mathsf{ipad})$ and $H(\bar{k} \oplus \mathsf{opad})$ to recover information about the key k. Thus our attack cannot be used against NMAC.

1.3 Related Work on Side Channel Attacks

Since there is no efficient and practical attacks against HMAC, it is interesting to study the security of this function against side channel attacks. Similarly, Kelsey *et al.* have studied the security of block ciphers using different leakage models [15,14].

A classical side channel attack on HMAC has been proposed without experiments by Lemke *et al.* in 2005 using a Differential Power Analysis [19] in [20]. They show that a forgery attack can be mounted by performing a multi-bit DPA since SHA-1 manipulates 32-bit registers. This attack allows to recover the inner and outer keys of HMAC, but does not allow to retrieve the initial key. Note that other DPA attacks are reported on HMAC based on other hash functions such as [22,10,21]. But none of them allow to retrieve the initial key value as the message is not directly mixed with the initial key but only with the derivated subkeys.

To our knowledge, no Template Attacks (TA) [5,25] have never been applied on HMAC. This is mainly due to the fact that the manipulated registers are 32 bits long which make classical templates attacks infeasible.

1.4 Our Results

The aim of this paper is two-fold. On the one hand, we assume that we have a side channel that leaks the number of flipped bits when a value is loaded from the memory to a register. We show that this side channel is sufficient to recover the secret key used in HMAC. On the other hand, we show that this side channel is available in practice. A similar attack can also be used against other standards or against key derivation process.

Our attack is quite different from previous side-channel attacks: we do not require knowledge of the message being signed, and we only need to measure *one* execution of HMAC. Thus some classical countermeasures against DPA will not affect our attack.

In our attack model, the adversary can profile the device during a first offline stage. This assumption is classical, and is used in template attacks. It seems reasonable since the adversary can buy and test its own device. During this profiling phase, the adversary generates curves when loading data from the memory to a register with all the possible Hamming distances. Then, in a second online stage, he has access to the targeted device during one execution of the implementation. During this phase, according to the recorded values, the Hamming distance can be learned using a single matching between the curves. Finally, all theses Hamming distances give some information about the key, and we can combine them to recover the key, even though the secret-dependent variables are not totally independent. The simulation of the attack has been experimentally tested to show that we are able to recover the secret key in practice and to show that the scale can be generated.

Our attack is based on two important facts. First, the key of HMAC is used as a message in the hash function. The message is processed word by word, as opposed to the IV which is used all at once, so we can extract information that depends on only one word of the key. Second, we have two related computations that leak information on the key. Note that it is important to have two different values of W_0: if HMAC were defined with $k||\mathsf{opad}$ and $k||\mathsf{ipad}$, the two series of measures in the first step would be exactly the same, and we would not have enough information for a practical attack.

The main difference between our attack and classical template attacks is that we do not consider the value of a register but its Hamming weight. This presents the advantage to limit the number of profiling (only 33 records are needed instead of 2^{32} for 32-bit registers) even if it gives less information. If the targeted register is manipulated a sufficient number of time, then we can combine the partial information we have recovered to find the entire value of the initial register. To sum up, this attack can be viewed as a combination of template attacks and classical power consumption model. Classical template attacks usually separate keys according to words or substrings of words. To our knowlegde and even if it seems natural, it is the first time that a template attack is based on power consumption models.

1.5 Organization of the Paper

In section 2, we describe SHA-1 and we present how our attack works. In section 3, we give experimental results on a real implementation on the NIOS processor embedded in a FPGA Altera Stratix. Finally, in section 4, we show that our HMAC attack can be applied to other hash functions such as MD5 and SHA2. We also show how a similar attack can be used against other constructions: it can break the confidentiality of the L2TP protocol and can recover the key materials of the IPsec protocol by attacking the key derivation function.

2 Description of the Attack

2.1 SHA-1 Message Leak

In SHA-1, the message is introduced word by word, as opposed to the IV which is introduced all at once. This means that the values manipulated during the first rounds of SHA-1 depend only on the first message words. For instance, the internal value A_1 depends only on the IV and on m_0. The value A_1 is also used as B_2 and after a rotation as C_3, D_4 and E_5. Each time this value is manipulated, there will be some leakage depending only on m_0. If we can model this leakage, we will be able to recover m_0: for all possible values of m_0, we simulate the leak, and we keep the values that give a good prediction. The full message recovery algorithm is given by Algorithm 1.

Algorithm 1. Recovery of n message words

```
 1: Profiling Stage
 2:     Study the device and model the leakage

 3: Operational Stage
 4:     Get side-channel information from one run of the hash function

 5: Message recovery
 6:     $\mathcal{S} \leftarrow \{\text{IV}\}$                    ▷ $\mathcal{S}$ is the candidate set
 7:     for $0 \le i < n$ do
 8:         $\mathcal{S}' \leftarrow \emptyset$
 9:         for all $s \in \mathcal{S}$ do
10:             for all $m_i \in \mathbb{Z}_{2^{32}}$ do
11:                 Simulate the leak for $(s, m_i)$
12:                 if it matches the measure then
13:                     $\mathcal{S}' \leftarrow \mathcal{S}' \cup (s, m_i)$
14:                 end if
15:             end for
16:         end for
17:         $\mathcal{S} \leftarrow \mathcal{S}'$
18:     end for
```

The complexity of the message recovery depends on the size of the set of good candidates $\mathcal{S}$: each iteration of the loop in line 7 costs $2^{32}|\mathcal{S}|$. If we have γ candidates matching each measure, the set $\mathcal{S}$ will be of size γ^i after iteration i, and the total cost of the algorithm to recover n message words is about $2^{32}\gamma^n$. The value of γ will depend of the number of information leaked through the side channel.

2.2 HMAC Key Leak

If we look at HMAC, we see that the secret key is used as a message for the hash function, so we can use the message leak in SHA-1 to recover the key of

HMAC-SHA-1. Note that this does not apply to NMAC, where the key is used as the initialization vector.

In fact, the secret key is used twice in the HMAC construction: it is used in the inner hash function as $H(\bar{k} \oplus \mathsf{ipad})$ and in the outer hash function as $H(\bar{k} \oplus \mathsf{opad})$. We can collect side-channel information from those two related messages, which gives two sets of measures for the same key. This property is crucial for our attack: with only one set of measures, we would have too many candidates in the set $\mathcal{S}$.

We will now study an implementation of SHA-1 on the NIOS processor to check whether this leakage is sufficient to identify the secret key.

2.3 Modelization of the Attack

The side channel we will use in our practical experiments is a measure of the electromagnetic radiation (see Section 3). We model it as follows: each time the processor loads a data into a register, the electromagnetic signal depends on the number of flipped bits inside this register.

More precisely, we target the `ldw` instruction (*load word*), which loads data from the memory to a register. Since this instruction does not perform any computation, we expect the EM signal to be quite clean, and we should be able to read the number of flipped bits. When a value B is loaded into a register whose previous value was A, we have a transition $A \to B$, and we can measure the Hamming weight of $A \oplus B$.

2.4 Study of an Implementation of SHA-1

For our experiments, we used the code of XySSL[1] which is an SSL library designed for embedded processor. The compiled code for the round function of SHA-1 is given in Table 1 in Appendix A, together with the values leaked from the `ldw` instructions.

Using this implementation of SHA-1, we have 6 measures at each step of the compression function, which gives 12 measures per key word of HMAC-SHA-1. The Hamming weight of a 32-bit value contains 3.54 bits of entropy, so we expect to have a small number of candidates for each key word. Note that the measures are clearly not independent, which makes it difficult to compute the number of expected candidates. Therefore, we ran some simulations to estimate the complexity of the attack.

2.5 Simulations

To estimate the complexity of the attack, we can run the message recovery step for some initial states. For a given state $(A_i, B_i, C_i, D_i, E_i)$, we consider the 2^{32} values of the message word W_i, and we compute the Hamming weight of the measures given in Table 1. We can count the number of collisions in

[1] Available at `http://xyssl.org/code/`

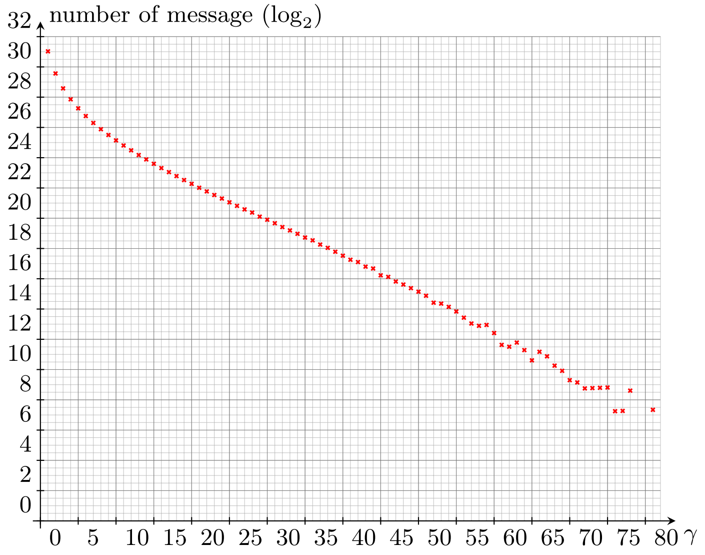

Fig. 1. Distribution of the number of candidates γ, for the first message word W_0

the measures, which will give the distribution of the number of candidates γ matching the measure.

More precisely for the first message word W_0, we know exactly the constants involved in the measures: they are given in Table 1, and we will use them with $W_0 = k_0 \oplus \mathsf{ipad}$ and $W_0 = k_0 \oplus \mathsf{opad}$. The distribution is given in Figure 1. It has an average of 2.79 and a worst case value of 81. Experiments with random initial state instead of the specified IV of SHA-1 give similar results (this simulates the recovery of the other message words). Moreover, if we allow one of the measures to be wrong with a difference of one, then we have 4.60 candidates on average, and 140 in the worst case. This means that we can tolerate some errors in the measures, without affecting too much the computation time.

In the end, we will assume that the measures give about three candidates at each step. We expect that the recovery of a 32κ-bit HMAC key will use a set $\mathcal{S}$ of size about 3^κ, and the complexity of the attack will be about $2^{32} \times 3^\kappa$ which will take a few hours on a PC for a 128-bit key. To identify the correct key among the set $\mathcal{S}$, we can either use a known MAC, or use some leakage in the following steps of the compression function.

3 Experimental Validation on a Known Implementation of HMAC-SHA1

Our side channel attack recovers the message input of the compression function if we can measure the number of bits flipped when loading from the memory to a register with the `ldw` instruction. We experimentally validated this assumption using an implementation of HMAC-SHA1 for a NIOS processor, on an Altera Stratix FPGA. The electromagnetic radiations are measured by near field methods using a loop probe sensitive to the horizontal magnetic field. We assume that the assembly code of is known, and we first study the implementation of the HMAC-SHA1 algorithm. Then, we focus on the leakage of the load instruction.

Electromagnetic radiation signals are now considered as one of the most powerful side channel signals [9,1,23]. One advantage of the EM side channel is that is possible to focus on local emanations.

3.1 Leakage of HMAC-SHA1 Implementation

The Figure 2 shows the radiations measured during the computation of HMAC-SHA1 with a 128-bit key and a 504-bit message M. We can clearly see the 5 executions of the compression function. A sharp analysis of this leakage needs to be done in order to find out some more useful information. During the experiments, we focus on the load instruction (referred as `ldw` by Stratix Assembly Language) but other instructions could also give information about the Hamming weight of the data used during the execution of SHA-1.

3.2 The Leakage of the `ldw` Instruction

The analysis in Section 2 shows that information leak from this instruction will be sufficient to recover the secret key. The goal of this section is to validate

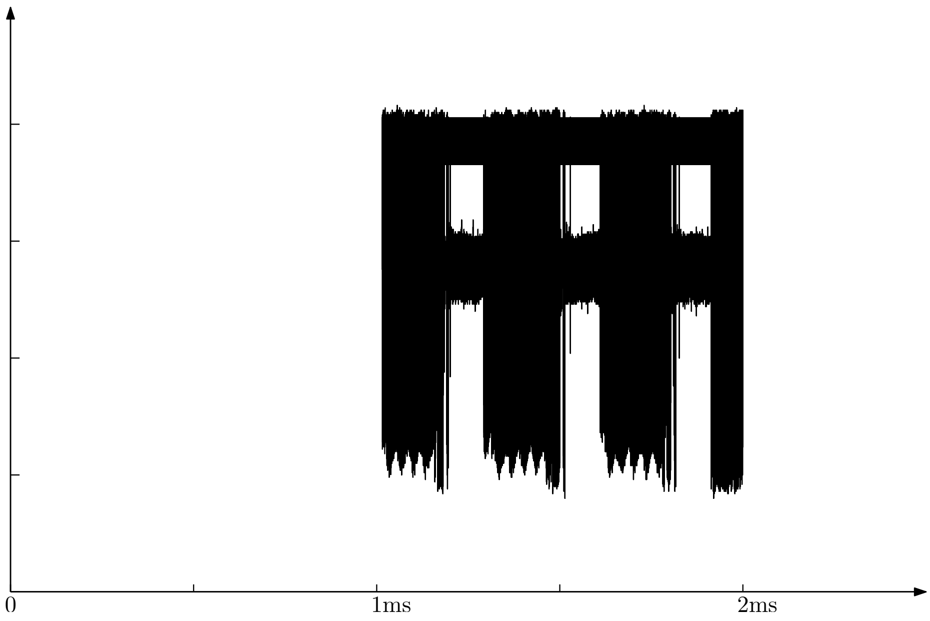

Fig. 2. HMAC-SHA1 Electromagnetic Execution. We see the three calls of the compression function to compute $h = H(\bar{k} \oplus \mathsf{ipad} \| M)$ and two calls to compute $H(\bar{k} \oplus \mathsf{opad} \| h)$.

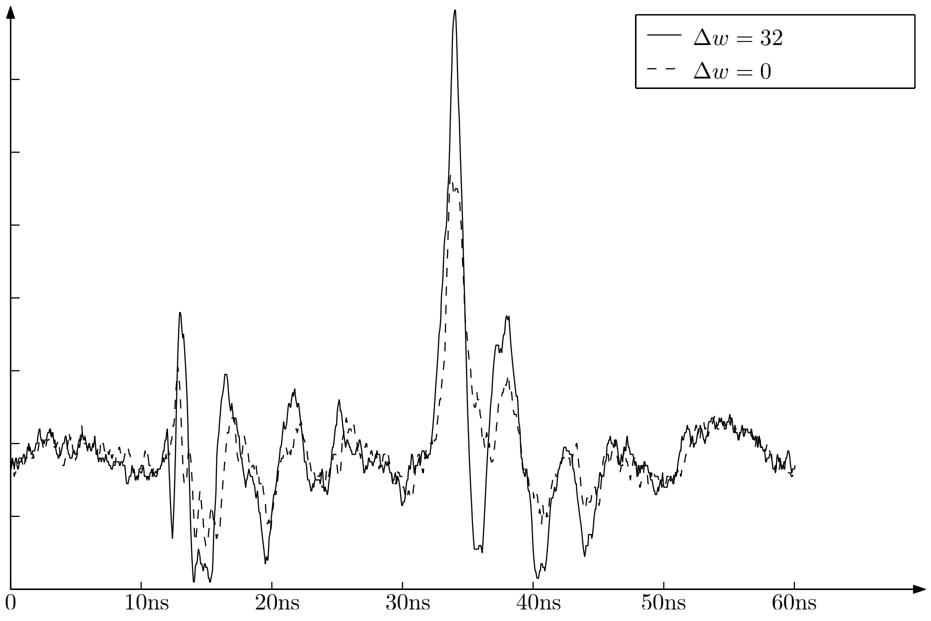

Fig. 3. Extremal Hamming Distances

that the Hamming distance between the mnemonic operands manipulated by the instruction `ldw` leak with electromagnetic radiations. As an example, if the register R, which value was A, is loaded with the new value B, we claim the Hamming distance between A and B leaks with the electromagnetic radiations.

For validation, we precharged the register R to A = `0xae8ceac8`, Altera Stratix being a 32-bit register technology. Then two experiments have been performed: in the first one, we use $B_1 = A$ = `0xae8ceac8` and in the second one, we use $B_2 = \bar{A}$ = `0x51731537`. These two experiments are opposite regarding the Hamming distance: $H(A \oplus B_1) = 0$ while $H(A \oplus B_2) = 32$. Fig. 3 illustrates this link between radiations and Hamming distance.

We must now check if the measures allow to distinguish a small difference in the Hamming weight, and if they are masked by the noise. For a successful attack, we need to be able to distinguish a Hamming distance of 15 from a Hamming distance of 16 since on average the frequency of this Hamming distance is larger than the extremal values. To verify this, we used pairs of values with those Hamming distance. Fig. 4 shows the results with the following pairs: (`0x00000000`, `0xe0f00ff0`), (`0x55555555`, `0x85a55aa5`), (`0x00000000`, `0xffff0000`), (`0xffffffff`, `0x0f0f0ff0`), (`0xaaaaaaaa`, `0x00050000`). We see that the curve depends on the Hamming distance between the two values, and not on the actual value of the register. Moreover, the noise level is sufficiently low in our setting to be able to distinguish a difference of one in the Hamming distance. These curves have been obtained by zooming on figure 3.

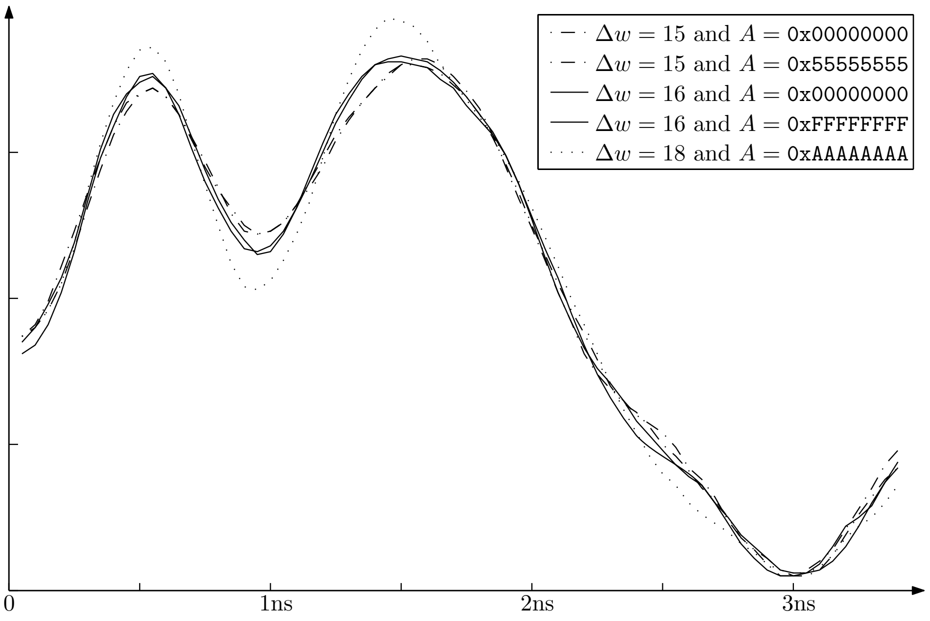

Fig. 4. Electromagnetic radiations for some Hamming distances

Thus, the Side Channel Analysis procedure can be done in two stages, a profiling stage and an operational stage. 33 measures of load instructions with all the possible Hamming distance are done during the profiling stage. This will allow us to find the Hamming distance of all `ldw` instructions for the operational stage. The profiling stage will also be used to study the timing of the SHA-1 computation, so as to match each instruction with the assembly code. Then, the operational stage consists in a Template Attack [5] on the `ldw` instructions. Following the attack of Section 2, we expect to recover a secret key of κ words with only one HMAC measure and a workload of about $2^{32}3^{\kappa}$ steps of the compressions function.

4 Extension to Other Hash Functions and to Other Usage

In this section, we show that the basic attack we proposed can be extended to other hash functions, works even though the code is unknown and can also be used to recover encryption keys in other protocols.

4.1 Other Hash Function of the MD4 Family

The other functions of the MD4 family (MD5, SHA-2, RIPEMD) have a very similar design and the message words also enter the compression function one by one. The assembly code of a specific implementation should be studied to see how many load instructions are used and what information is leaked, but we expect the attack to be just as efficient. Basically, our attack should work against any hash function based on a Feistel ladder.

4.2 Unknown Implementation

Even if we don't have access to the assembly code of the implementation, our attack is still applicable. The main difference with the previous analysis is that previous value of the targeted register A_{init} is unknown. Anyway, the attacker can improve the profiling stage. Indeed, he can guess the value of A_{init} with Correlation Power Analysis(CPA) [4], making the secret key varying. Let's remark that instead of making a 32 bits CPA, the attacker can do 4 CPA, each one on 8 bits. This procedure permits to limit the computational cost of the profiling stage. Then, thanks to this CPA procedure, the attacker can guess what was a register value before the secret key bits are written on it. Furthermore, the CPA peak permits to localize in time the load instruction.

With all these templates obtain at a very low computational cost, the attacker will be able with only one curve (if the noise is as low as in our setting) to retrieve the full key.

4.3 Other Attack Scenarios

In this section we identify some other construction where our attack can recover a secret key. The basic requirement is that the secret key is used as the message input of the hash function, and we need two related computations where the key enters the hash function in two different states.

Confidentiality: The L2TP Example. The L2TP protocol [27] is a tunneling protocol used to implement VPNs. It uses a shared secret K, and two known values RV and AV. The plaintext is broken into 16-byte chunks, p_1, p_2, ... The ciphertext blocks are called $c_1, c_2, \ldots$ and the intermediate values $b_1, b_2, \ldots$ are computed as follows:

$$\begin{array}{ll} b_1 = MD5(AV\|K\|RV) & c_1 = p_1 \oplus b_1 \\ b_2 = MD5(K\|c_1) & c_2 = p_2 \oplus b_2 \\ b_3 = MD5(K\|c_2) & c_3 = p_3 \oplus b_3 \ldots \end{array}$$

The secret key K enters the hash function in two different states for the computation of b_1 and b_2, so we can apply our attack and recover the key.

Key Derivation. Key derivation is sometimes used as a countermeasure against DPA attacks. The key derivation process described in [17], uses a hash function H, a master key K and a counter ctr, and computes the sessions keys as $SK = H(ctr\|K)$. Using our attack, if we observe only two key derivation process, we have enough information to recover the master key K.

Note About RIPEMD. The RIPEMD family of hash function uses two parallel Feistel ladder, and combines the results in the end of the compression function. This allows us to recover two series of measures, even if the secret key enters the hash function only once. The original function RIPEMD-0 uses two lines with the same permutation and different constants, which gives enough information for our attack. Thus, any construction which uses a secret key as a part of a message to be hashed with RIPEMD-0 is vulnerable. The newer functions RIPEMD-128 and RIPEMD-160 uses different permutations of the message in the two lines; our attack can reduce the key-space but we don't have a practical key recovery with a single hash function call.

4.4 Possible Countermeasure

A possible countermeasure against our attack is to keep the internal state of SHA-1 inside the processor registers. Our attack uses the fact that the internal state is stored in the stack and moved in and out the registers: we measure the radiations during this movement. If all the computations are done inside the registers, we can still measure the radiations during the computations, but the signal will have more noise. Another solution is to load random values between each ldw instruction or to use classical masking methods but this requires a random generator and may downgrade the performance drastically.

5 Conclusion

In this paper, we show that the electromagnetic radiation of a device can leak the number of flipped bits when data is loaded into a register. This information could also be observed using a proper current power analysis. However, EM signal allows to obtain emanations of local instructions and attacks are not intrusive since they can be performed in the near field of the device and do not require to modify the circuitry. Our experimentation studies the `ldw` instruction since it is easier to characterize during the profiling stage, but the attack could be improved by using other instructions. Our attack targets the message input of the compression function, while previous attacks only considered the IV input. This allows us to attack other standards and to recover crucial information such as the master key in some key-derivation schemes.

Finally, these results give some information about the efficiency of side channel attack on hash functions implemented on embedded processors. It is important to see that the adversary model is very limited: access to a similar device for a profiling stage and then one execution leads to an efficient key recovery.

References

1. Agrawal, D., Archambeault, B., Rao, J.R., Rohatgi, P.: The em side-channel(s). In: Kaliski Jr., B.S., et al [12], pp. 29–45
2. Bellare, M.: New proofs for NMAC and HMAC: Security without collision-resistance. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 602–619. Springer, Heidelberg (2006)
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. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Joye and Quisquater [11], pp. 16–29
5. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: Kaliski Jr., B.S., et al [12], pp. 13–28
6. Contini, S., Yin, Y.L.: Forgery and partial key-recovery attacks on hmac and nmac using hash collisions. In: Lai, X., Chen, K. (eds.) ASIACRYPT 2006. LNCS, vol. 4284, pp. 37–53. Springer, Heidelberg (2006)
7. Cramer, R. (ed.): EUROCRYPT 2005. LNCS, vol. 3494. Springer, Heidelberg (2005)
8. Fouque, P.-A., Leurent, G., Nguyen, P.Q.: Full key-recovery attacks on hmac/nmac-md4 and nmac-md5. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 13–30. Springer, Heidelberg (2007)
9. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic analysis: Concrete results. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 251–261. Springer, Heidelberg (2001)
10. 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)
11. Joye, M., Quisquater, J.-J. (eds.): CHES 2004. LNCS, vol. 3156. Springer, Heidelberg (2004)
12. Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.): CHES 2002. LNCS, vol. 2523. Springer, Heidelberg (2003)
13. Kaufman, C.: Rfc 4306 - internet key exchange (ike v2) protocol (December 2005), `http://www.ietf.org/rfc/rfc4306.txt`
14. Kelsey, J., Schneier, B., Wagner, D., Hall, C.: Side channel cryptanalysis of product ciphers. In: Quisquater, J.-J., Deswarte, Y., Meadows, C., Gollmann, D. (eds.) ESORICS 1998. LNCS, vol. 1485, pp. 97–110. Springer, Heidelberg (1998)
15. Kelsey, J., Schneier, B., Wagner, D., Hall, C.: Side channel cryptanalysis of product ciphers. Journal of Computer Security 8(2/3) (2000)
16. Kent, S.: Security architecture for the internet protocol (November 1998), `http://www.ietf.org/rfc/rfc2401.txt`
17. Kocher, P.: Us patent no. 6,304,658 (2003), `http://www.cryptography.com/technology/dpa/Patent6304658.pdf`
18. Kocher, P.: Us patent no. 6,539,092 (2003), `http://www.cryptography.com/technology/dpa/Patent6539092.pdf`
19. 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)
20. 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 and Quisquater [11], pp. 205–219
21. McEvoy, R.P., 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)
22. 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)

23. Quisquater, J.-J., Samyde, D.: Electromagnetic analysis (ema): Measures and counter-measures for smart cards. In: Attali, S., Jensen, T. (eds.) E-smart 2001. LNCS, vol. 2140, pp. 200–210. Springer, Heidelberg (2001)
24. Rechberger, C., Rijmen, V.: On authentication with hmac and non-random properties. In: Dietrich, S., Dhamija, R. (eds.) FC 2007 and USEC 2007. LNCS, vol. 4886, pp. 119–133. Springer, Heidelberg (2007)
25. Schindler, W., Lemke, K., Paar, C.: A stochastic model for differential side channel cryptanalysis. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 30–46. Springer, Heidelberg (2005)
26. Shoup, V. (ed.): CRYPTO 2005. LNCS, vol. 3621. Springer, Heidelberg (2005)
27. Towsley, W., Valencia, A., Rubens, A., Pall, G., Zorn, G., Palter, B.: Rfc 2661 - layer two tunneling protocol "l2tp" (August 1999), `http://www.ietf.org/rfc/rf2661.txt`
28. Wang, X., Lai, X., Feng, D., Chen, H., Yu, X.: Cryptanalysis of the hash functions md4 and ripemd. In: Cramer [7], pp. 1–18
29. Wang, X., Yin, Y.L., Yu, H.: Finding collisions in the full sha-1. In: Shoup [26], pp. 17–36
30. Wang, X., Yu, H.: How to break md5 and other hash functions. In: Cramer [7], pp. 19–35
31. Wang, X., Yu, H., Yin, Y.L.: Efficient collision search attacks on sha-0. In: Shoup [26], pp. 1–16

A SHA-1 Code

Table 1. SHA-1 code. The table shows the code for one step of the compression function. The other steps are exactly the same, excepted that the mapping between the internal state values A, B, C, D, E and the stack changes at each round.

	Stack					*Registers*			*Measure*
Instruction	76	80	84	88	92	`r2`	`r3`	`r4`	
Begin step 0	A_0	B_0	C_0	D_0	E_0	X_0	Y_0	Z_0	
`ldw r2 76(fp)`						A_0			$X_0 \rightarrow A_0$
`roli r4, r2, 5`								$A_0^{\lll 5}$	
`ldw r3, 84(fp)`							C_0		$Y_0 \rightarrow C_0$
`ldw r2, 88(fp)`						D_0			$A_0 \rightarrow D_0$
`xor r3, r3, r2`							$C_0 \oplus D_0$		
`ldw r2, 80(fp)`						B_0			$D_0 \rightarrow B_0$
`and r3, r3, r2`							$(C_0 \oplus D_0) \wedge B_0$		
`ldw r2, 88(fp)`						D_0			$B_0 \rightarrow D_0$
`xor r2, r3, r2`						Φ_0			
`add r3, r4, r2`							$\Phi_0 \boxplus A_0^{\lll 5}$		
`ldw r2, 12(fp)`						W_0			$\Phi_0 \rightarrow W_0$
`add r3, r3, r2`							$\Phi_0 \boxplus A_0^{\lll 5} \boxplus W_0$		
`ldw r2, 92(fp)`						E_0			$W_0 \rightarrow E_0$
`add r3, r3, r2`							$\Phi_0 \boxplus A_0^{\lll 5} \boxplus W_0 \boxplus E_0$		
`movhi r2, 23170`						`0x5a820000`			
`addi r2, r2, 31129`						`0x5a827999`			
`add r2, r3, r2`						A_1			
`stw r2, 92(fp)`					A_1				
`ldw r2, 80(fp)`						B_0			$A_1 \rightarrow B_0$
`roli r2, r2, 30`						$B_0^{\lll 30}$			
`stw r2, 80(fp)`		C_1							
Begin step 1	B_1	C_1	D_1	E_1	A_1	X_1	Y_1	Z_1	
`ldw r2, 92(fp)`						A_1			$X_1 \rightarrow A_1$
...									

We have the following relations:

$$
\begin{aligned}
\Phi_i &= f_i(B_i, C_i, D_i) \\
B_{i+1} &= A_i \\
C_{i+1} &= B_i^{\lll 30} \\
D_{i+1} &= C_i \\
E_{i+1} &= D_i \\
A_{i+1} &= \Phi_i \boxplus A_i^{\lll 5} \boxplus W_i \boxplus E_i \boxplus K_i \\
X_{i+1} &= B_i^{\lll 30} \\
Y_{i+1} &= \Phi_i \boxplus A_i^{\lll 5} \boxplus W_i \boxplus E_i \\
Z_{i+1} &= A_i^{\lll 5}
\end{aligned}
$$

We can make 8 measures per step, but this gives only 6 informations leaks, because the transitions $D_{i+1} \rightarrow B_{i+1}$ and $B_{i+1} \rightarrow D_{i+1}$ leaks the same information as $X_i \rightarrow A_i$.
For instance, the leaks related to W_0 are:

$\Phi_0 \rightarrow W_0$	$W_0 \oplus$ `0x98badcfe`
$W_0 \rightarrow E_0$	$W_0 \oplus$ `0xc3d2e1f0`
$A_1 \rightarrow B_0$	$(W_0 \boxplus$ `0x9fb498b3`$) \oplus$ `0xefcdab89`
$X_1 \rightarrow A_1$	$(W_0 \boxplus$ `0x9fb498b3`$) \oplus$ `0x7bf36ae2`
$Y_1 \rightarrow C_1$	$(W_0 \boxplus$ `0x45321f1a`$) \oplus$ `0x7bf36ae2`
$A_1 \rightarrow D_1$	$(W_0 \boxplus$ `0x9fb498b3`$) \oplus$ `0x98badcfe`