IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2023, No. 3, pp. 1–29. DOI:10.46586/tches.v2023.i3.1-29

Carry-based Differential Power Analysis (CDPA) and its Application to Attacking HMAC-SHA-2

Yaacov Belenky, Ira Dushar, Valery Teper, Vadim Bugaenko, Oleg Karavaev, Leonid Azriel and Yury Kreimer

FortifyIQ, Inc., 300 Washington Street, Suite 850, Newton, MA 02458 USA
{belenky,dushar,teper,bugaenko,karavaev,azriel,kreimer}@fortifyiq.com
https://www.fortifyiq.com/

Abstract. In this paper, we introduce Carry-based Differential Power Analysis (CDPA), a novel methodology that allows for attacking schemes that use arithmetical addition. We apply this methodology to attacking HMAC-SHA-2. We provide full mathematical analysis of the method and show that under certain assumptions and with a sufficient amount of traces any key can be revealed. In the experimental part of the paper, we demonstrate successful application of the attack both in software simulation and on an FPGA board using power consumption measurements. With as few as 30K traces measured on the FPGA board, we recover the secrets that allow for forging the HMAC-SHA-2 signature of any message in 3% of the cases — while with 275K traces the success rate reaches 100%. This means that any implementation of HMAC-SHA-2, even in pure parallel hardware, is vulnerable to side-channel attacks, unless it is adequately protected. To the best of our knowledge, this is the first published full-fledged attack on pure hardware implementations of HMAC-SHA-2, which does not require a profiling stage.

Keywords: Side-channel analysis · DPA · HMAC · SHA-2 · SHA-256

1 Introduction

The Keyed-Hash Message Authentication Code (HMAC) was standardized by NIST in FIPS PUB 198-1 [Nat08], and since its standardization, it is widely used for symmetric message authentication. HMAC is constructed using an approved hash function, namely SHA-1 and several functions of the SHA-2 family [Nat12]. Recently, NIST standardized the SHA-3 hash function family [Nat15]. Nevertheless, HMAC-SHA-2 still arguably remains the most popular variant of HMAC.

This level of ubiquitousness inevitably attracts the interest of the research community in the sense of applying various sorts of attacks to break the scheme. For example, side-channel attacks, and power analysis attacks in particular [KJJ99, Koc96] are a very powerful tool capable of uncovering secrets in virtually any insufficiently protected cryptographic scheme. Numerous side-channel attacks on AES, RSA, ECC, Diffie-Hellman and other algorithms have been published. Surprisingly, until recently, no successful attacks on HMAC-SHA-2 implemented in pure parallel hardware have been presented, which has resulted in little attention to HMAC in the area of vulnerability analysis [BSI13].

Several publications on side-channel attacks on HMAC-SHA-1 and HMAC-SHA-2 either provided only partial analysis without showing the full path [MTMM07, RM13], or were applicable only to software [FLRV09, BBD+15, KGB+18], or used an unrealistic chosen input data assumption [GWM16], or attacked exotic usages or error-prone implementations [APSQ06, Osw16]. More detailed analysis of the prior work can be found

Received: 2023-01-15 Accepted: 2023-03-15 Published: 2023-06-09

in [BDT+21]. Some of the publications were missing experimental data, some have shown only correlations in the experiments, but not an actual attack. None of the aforementioned publications have shown a full path of the attack applied to a real hardware implementation.

We can name several reasons for the apparent resilience of HMAC against side-channel attacks. The first reason lies in the HMAC design, which involves two invocations of its underlying hash function on the secret key K (1), called "inner hash" and "outer hash". Even if the adversary has full control over the input data and manages to break the inner hash, he can discover the input to the outer hash, albeit he still cannot choose it as he wishes, which severely limits his possibilities. HMAC is defined as

$$HMAC(K, M) = \overbrace{H\left(\left(K_0 \oplus opad\right) \| H\overbrace{\left((K_0 \oplus ipad) \| M\right)}^{\text{inner hash}}\right)}^{\text{outer hash}} \tag{1}$$

where H is an approved hash function, K_0 is a known function of the secret key K, M is the input message, and *ipad* and *opad* are known constants.

The second reason stems from the construction of the SHA-1 and SHA-2 functions, which involves arithmetic addition. Side-channel attacks benefit from substantial leakage as a result of even small changes in input data. This allows for using a small hypothesis space with good separation. Hence, functions that amplify small changes in the input present an easy target for these attacks. S-boxes, heavily used in block ciphers, are a perfect target in this sense, since a change of a single bit of the input to the S-box changes many bits of its output. In contrast, the XOR function is a difficult target, since it provides no amplification at all — a change in one bit of the input causes a change only in the same bit of the output. Arithmetical addition is similar to XOR with only slightly better average amplification — a one-bit change in the addend in average results in two or less bit changes in the sum.

In general, it is more difficult to exploit the leakage of a function if it has low amplification. However, limiting the analysis to one- or two-bit windows, while averaging away the other bits, may provide the desired result. Belenky *et al.* [BDT+21] harnessed the narrow bit window approach to mount a practical template attack on hardware implementations of HMAC-SHA-2. The full attack path was validated on an FPGA board. The suggested attack does not rely on an analytical model, but rather uses a brute-force approach to build the template tables. Hence, a substantial amount of traces both in the profiling stage and in the attack stage is required to extract the secret. In addition, there are typical assumptions for the template attacks, such as access to an open device for the profiling stage.

In this paper, we introduce Carry-based Differential Power Analysis (CDPA), a new technique for differential power analysis on cryptographic schemes that use arithmetical addition. [1]

The attack works in the Hamming distance model, and the target of the attack is an arithmetical addition of a secret addend to a known addend that replaces a secret previous value in the target register. We recover the secret addend and the secret previous value in the target register bit by bit, from the LSB to the MSB. The hypotheses are based on the carry bit from the current bit position into the next bit. Namely, in step i, the $i-1$ previous bits of the secret addend are already known from the previous steps, and the goal is to find bit i. Using the knowledge of the secret addend bits, we split the set of the possible values of the known addend into two pairs of subsets such that the average difference of the Hamming distances between the two subsets in each pair depends only on the carry into the target bit i. We estimate this average difference using the set of Hamming distances corresponding to a sufficiently large set of randomly distributed values

[1] We use the term DPA in its broad sense, so it includes various acquisition techniques, such as measuring the supply current or electromagnetic (EM) radiation.

of the known addend, and find the border value at which the sign of the difference switches due to the change in the carry to the target bit.

In the case of attacking HMAC-SHA-2, two such additions simultaneously, rather than one, pose an additional challenge. In each bit position, we need to find two, rather than one, border values, at which the two carry bits change. Whenever the two values coincide or are close one to another, we split the set of traces in certain cases into more than four equal subsets, and in other cases into 8 unequal subsets. Some or all of these subsets may be too small, so that a larger amount of traces will be required for the attack. The number of traces necessary to recover the secret values depends on the relationship between the secret addends; generally, the longer the sequences of the matching bits, the higher the number of the required traces. Nevertheless, any given secret can be eventually revealed, using a sufficient amount of traces under the Hamming distance leakage model.

In the real world, side-channel leakage includes noise in addition to the part of the leakage proportional to the Hamming distance. In particular, it includes noise from the operations in the combinational logic, which involve the known addend, and therefore this kind of noise is correlated with the Hamming distance on the registers. Although theoretically, such noise may render the attack impossible, in our experiments based on power consumption measurements with a randomly chosen key, the attack had significant success rates, from 3% at 30K traces to 100% at 275K traces. We believe that these results can be further improved by measuring EM radiation using a properly positioned probe.

To summarize, we make the following contributions in this paper:

1. We present Carry-based Differential Power Analysis (CDPA) — a new category of differential power analysis intended for attacking algorithms involving arithmetical addition in the Hamming distance leakage model.

2. We present an attack on HMAC-SHA-2 — an enhancement of the basic CDPA, which covers two simultaneous addition operations. To the best of our knowledge, this is the first practical full-fledged attack on a pure parallel hardware implementation of HMAC-SHA-2 that does not require a profiling stage.

3. We show analytically that for both the basic CDPA and the attack on HMAC-SHA-2 any secret can be revealed given a sufficient amount of traces and assuming no correlated noise.

4. We demonstrate the attack on an FPGA board, achieving success already with as little as 30K traces (in 3% of the cases), which is less than the state of the art [BDT+21] by about an order of magnitude and does not require a profiling stage.

The remainder of this article is organized as follows. Section 2 describes the basic construction of CDPA. Section 3 describes the enhanced three-stage version of CDPA which is able to attack HMAC-SHA-2. Section 4 discusses the real world setting with correlated noise from combinational logic, and the fine-tuning of the attack on HMAC-SHA-2 for this setting. Section 5 presents the experimental results, both in simulation and on an FPGA board. Section 6 presents our conclusions.

2 Carry-based Differential Power Analysis

2.1 Notation

$\oplus$ means XOR.
$X[j]$ means the bit number j of an integer X, where index 0 corresponds to the least significant bit.
$X[j:k]$ means the binary number represented by the bits $[j:k]$ of an integer X (from the most significant to the least significant) if $j \geq k$, and 0 if $j < k$.

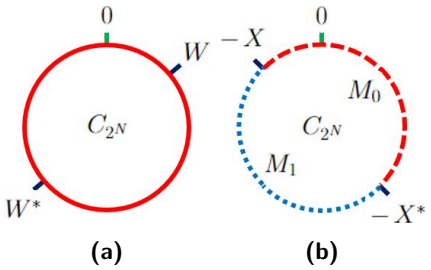

Figure 1

By definition,

$$HD(X,Y) = \sum (X[i] \oplus Y[i]) \tag{2}$$

(the Hamming distance between integers X and Y), and

$$HD_{[j:k]}(X,Y) = \sum_{i=k}^{j} (X[i] \oplus Y[i]) \tag{3}$$

2.2 The Statement of the Problem

A device performs arithmetic addition $X + W$, where X is an N-bit secret value, and W is a known N-bit input. The N least significant bits of the result overwrite a register, containing another N-bit secret value, Y. Since all the arithmetic is N-bit, *i.e.*, modulo 2^N, we'll identify the N-bit numbers with the elements of the cyclic additive group C_{2^N}, and the addition and subtraction below are in the sense of this group.

For any input value W, the attacker obtains the Hamming distance

$$L_{X,Y}(W) = HD(X + W, Y) \tag{4}$$

between the two states of the register as a side-channel leakage. The problem to be solved is: Using multiple experiments with known values of the input word W whose bits are distributed independently and uniformly, find the secret values X and Y. (In $L_{X,Y}(W)$ we will drop the indices X, Y when they are implied.)

2.3 Solution

2.3.1 The Main Observation.

The solution is based on one simple observation. Let's depict C_{2^N} as a circle. For any $W \in C_{2^N}$ let

$$W^* = W + 2^{N-1} \tag{5}$$

be the opposite point on the circle (see Figure 1a).

Clearly, W and W^* differ only in the most significant bit, and so do $X + W$ and $X + W^*$. Clearly, $W^{**} = W$.

We denote:

$$L^*(W) = L(W^*) \tag{6}$$

and

$$\partial L(W) = L^*(W) - L(W) \tag{7}$$

It is easy to see that:

1. $\partial L(W^*) = -\partial L(W)$.

2. $\partial L(W) = \pm 1$ — more specifically, $\partial L(W) = 1$ if the most significant bits of $X + W$ and Y coincide, otherwise $\partial L(W) = -1$.

3. There are exactly two points on the circle in which $\partial L(W)$ switches its sign — when W changes from $-X - 1$ to $-X$, and when W changes from $-X^* - 1$ to $-X^*$ (see Figure 1b).

Denoting

$$\partial^2 L(W) = \partial L(W) - \partial L(W - 1) \tag{8}$$

we can equivalently say that the function $\partial^2 L(W)$ is 0 everywhere except at the two points $-X$ and $-X^*$, where it assumes values ± 2.

Assuming for a moment that the attacker is allowed to choose W as he wants, he can evaluate $\partial L(W)$ for all the values of W [2], find the pair of points at which $\partial L(W)$ changes its sign, and deduce the pair $\langle X, X^* \rangle$, or equivalently, deduce

$$T = X[N - 2 : 0] \tag{9}$$

Moreover, it is easy to see that

$$\partial L(0) = (1 \oplus X[N-1] \oplus Y[N-1]) - (X[N-1] \oplus Y[N-1]) = \\ = \begin{cases} +1 & \text{if } X[N-1] \oplus Y[N-1] = 0 \\ -1 & \text{if } X[N-1] \oplus Y[N-1] = 1 \end{cases} \tag{10}$$

so it is possible to deduce $X[N-1] \oplus Y[N-1]$ as well:

$$X[N-1] \oplus Y[N-1] = \begin{cases} 0 & \text{if } \partial L(0) = +1 \\ 1 & \text{if } \partial L(0) = -1 \end{cases} \tag{11}$$

or equivalently

$$X[N-1] \oplus Y[N-1] = \begin{cases} 0 & \text{if } \partial^2 L(T) = -2 \\ 1 & \text{if } \partial^2 L(T) = +2 \end{cases} \tag{12}$$

where T is defined by (9).

2.3.2 What is Still Missing?

The above method is a step towards the solution. However, several points are still missing.

1. We cannot find $X[N-1]$, but only $X[N-1] \oplus Y[N-1]$. This is in fact inevitable, because a simultaneous flip of $X[N-1]$ and $Y[N-1]$ does not affect $L(W)$.

2. We found no bits of Y. We'll return to this later.

3. Most importantly, the problem as stated assumes known, rather than chosen, bit-wise uniformly distributed values of W.

[2] It is possible to achieve the same goal by evaluating $\partial L(W)$ at N points only. We will not explore this direction, since in this section we describe only the basic observation, not the actual method that we suggest to use for the attack.

2.3.3 How to Drop the Assumption of a Chosen W?

In order to drop the assumption of a chosen W, we need several additional observations. Before we list them, we introduce some useful definitions.

Definition 1. For natural numbers N and k, a function $F : C_{2^N} \to \mathbb{R}$ is called a *k-step function* if C_{2^N}, seen as a cyclic sequence, can be split into k intervals (not necessarily of equal sizes) such that in each interval the function F is constant.

Definition 2. For natural numbers N and k, a function $F : C_{2^N} \to \mathbb{R}$ is called a *k-peak function* if it is different from 0 in at most k points.

Note that in both Definition 1 and Definition 2 we do not demand that k be the minimal number with one of the above properties. Therefore if $k < n$ then any k-step (k-peak) function is also an n-step (n-peak) function.

Definition 3. For a natural number N, a function $F : C_{2^N} \to \mathbb{R}$ is called *odd* if

$$\forall (W \in C_{2^N}) \big(F(W) = -F(W^*)\big)$$

Definition 4. For $F : C_{2^N} \to \mathbb{R}$ and $M \subset C_{2^N}$, $F(M)$ is by definition the average value of F over the subset M.

In these definitions, $\partial L(W)$ is an odd 2-step function, and $\partial^2 L(W)$ is an odd 2-peak function.

Now, the additional observations.

1. If $\partial L(W)$ is guaranteed to be constant in an interval M of C_{2^N}, then

$$\forall (W \in M) \big(\partial L(W) = \partial L(M) = L^*(M) - L(M)\big) \tag{13}$$

2. It is possible to analyze the addition, limited to the i least significant bits ($i < N$), modulo 2^i in the same way as we analyze the full N-bit addition modulo 2^N.

3. If $T = X[i-2:0]$ is known (Figure 2a), then modulo 2^{i+1} there are four intervals M_0, M_1, M_2, M_3 corresponding to different values of $(T+W)[i:i-1]$, at which the 2-peak function $\partial L(W)$ (modulo 2^{i+1}) is guaranteed to be constant, with two options for the pair of points where the sign of $\partial L(W)$ actually changes, as shown in Figure 2b and Figure 2c.

4. Since the definition of the sets M_k depends only on the bits $(T+W)[i:i-1]$, and since the bits of W are by assumption distributed uniformly and independently, the average value of $(X+W)[k] \oplus Y[k]$, where $k \neq i,\ k \neq i-1$, in any one of these subsets is close to 0.5, and its deviation from 0.5 decreases inversely proportionally to the square root of the subset size. Additionally, in each pair of the opposite sets (M_0 and M_2, M_1 and M_3) the bit $(T+W)[i-1]$ has identically the same value. Therefore when estimating $\partial L(M_k)$ based on the experimental data, all the terms except for the one corresponding to the bit position i, cancel out asymptotically, and the total deviation decreases inversely proportionally to the square root of the subset size. Therefore for a sufficiently large set of values of W this total error is small enough for the attacker to be able to find out whether the value is $+1$ or -1.

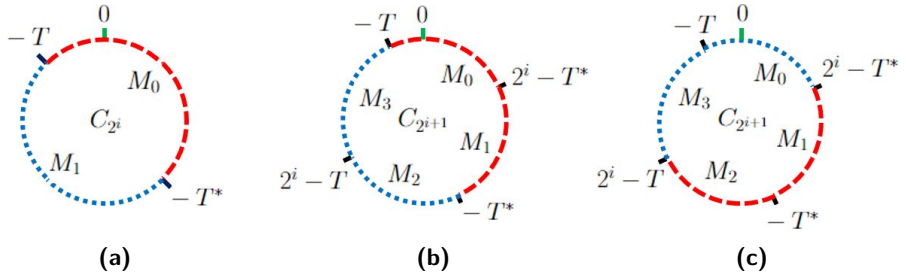

Figure 2

2.3.4 Putting It All Together

Based on all the observations, the practical attack with known input W is performed in steps numbered from 1 to $N-1$, in ascending order, and works as follows.

The prerequisite for step i is the knowledge of $T = X[i-2:0]$. (In particular, for $i=1$ it means that there are no prerequisites.) The analysis is modulo 2^{i+1}. We split the set of the experiments into four subsets M_k $(0 \le k < 4)$ as shown in Figure 2, and estimate $L(M_k)$ by averaging $L(W)$ over a large uniformly distributed subset of M_k. Then from the pair of points T', T'^* in which $\partial L(M_k)$ changes its sign we deduce $X[i-1]$, thus ensuring the prerequisite for step $i+1$. Additionally, from the direction of this change (in other words, from the sign of $\partial^2 L(T')$) we deduce $X[i] \oplus Y[i]$, according to (12).

After all these steps, we know $X[N-2:0]$ and $X[i] \oplus Y[i]$ for $0 < i < N$. From this $Y[N-2:1]$ is easily calculated.

The only missing bit is now $Y[0]$. In order to find it, we perform an additional step which we call step 0 — the analysis modulo 2, and deduce $X[0] \oplus Y[0]$ from the sign of $\partial L(M_0) = L(M_1) - L(M_0)$, according to (11). Since $X[0]$ is already known, we can calculate $Y[0]$ as well.

Step 0 is independent of all other steps and can be performed at any point of time.

After all the steps, we know $X[N-2:0]$, $Y[N-2:0]$ and $X[N-1] \oplus Y[N-1]$, or equivalently two hypotheses regarding $\langle X, Y \rangle$, corresponding to the two possible values of $X[N-1]$. As already mentioned, it is impossible to find more than this in the Hamming distance leakage model.

2.3.5 CDPA of Higher Orders

Table 1: The number of traces for success rate $> 50\%$

	32-bit words			64-bit words		
Noise	1st order	2nd order	3rd order	1st order	2nd order	3rd order
0	2^9	2^{15}	$> 2^{20}$	2^{10}	2^{17}	$> 2^{20}$
4	2^{10}	2^{17}		2^{11}	2^{18}	
8	2^{12}	2^{20}		2^{13}	2^{20}	
16	2^{14}			2^{14}		
32	2^{16}			2^{16}		
64	2^{18}			2^{18}		
128	2^{20}			2^{20}		

The CDPA attack described in the previous sections assumes an unprotected implementation. In this section we will study the case of a protected implementation, in which each word is represented as an XOR of several shares. Similarly to the previous sections, we will assume that the side channel leakage is the Hamming distance between representations

of $X + W$ and of Y in n shares both. If the number of shares is greater than 1, then the average Hamming distance does not depend on the values of X, Y, and W, so CDPA as described above does not work. However, it is possible to perform a similar attack based on higher moments of the distributions of the Hamming distances in the same subsets M_i instead of being based on their averages.

In order to find out whether the attack actually works in higher orders, we wrote a Python script `test_cdpa_attack.py` which is publicly accessible at https://github.com/fortify-iq/cdpa/tree/master/src/. The script randomly generates secret words X and Y and a list of known words W_i. For each word W_i it randomly represents $X + W_i$ and Y in n shares each, and calculates the corresponding side channel leakage value as the Hamming distance between the shares of $X + W_i$ and of Y, optionally adding to it normally distributed noise. The number of traces, the number of shares and the standard deviation of the noise are command line parameters of the script. In order to attack an implementation in n shares, the script uses the n^{th} moments instead of the averages.

Using this script, we studied several success metrics as a function of:

- The word size (32 and 64 bits);
- The number of shares (1, 2 and 3);
- The amplitude of noise (0, 4, 8, 16, 32, 64, 128).

We used the following success metrics:

- The success rate of the attacks (M_1);
- The percentage of the correctly found pairs of bits of X and Y, up to the least significant incorrect bit in each attack attempt (M_2);
- The percentage of all the correctly found bits (M_3).

An Excel file that presents the full outcome of this study can be found at https://github.com/fortify-iq/cdpa/blob/master/docs/cdpa_stats.xlsx.

A short summary of the results is presented in Table 1. For every combination of the word size, number of shares, and noise amplitude, it shows the minimal number of traces which is a degree of 2 for which $M_1 > 50\%$.

The three metrics used proved to be highly correlated, so changing the condition $M_1 > 50\%$ to the condition $M_2 > 65\%$ or $M_3 > 95\%$ would produce exactly the same table.

Our conclusions from this study:

1. First order CDPA is easy (starting from hundreds of traces) and not very sensitive to noise.
2. Second order CDPA is more difficult than first order CDPA by about two orders of magnitude (starting from tens of thousands of traces) and is more sensitive to noise and to the word size. Still, it may be practical.
3. Third order CDPA theoretically works. With 1M traces and without noise it gives about 90% correctly guessed bits and about 17% correctly guessed 32-bit words. However the number of traces required for a non-negligible success ratio is larger than 500K even without any noise, so its applicability to real world devices is dubious.

		Step 0	Step 1	Step 2	Step 3	Step 4	Step 5	Step 6	Step 7
W	HD	W&1	((W+ 0) >>0)&3	((W+ 1) >>1)&3	((W+ 1) >>2)&3	((W+ 5) >>3)&3	((W+ d) >>4)&3	((W+ d) >>5)&3	((W+ d) >>6)&3
27	5	M1	M3	M0	M2	M1	M3	M1	M0
c8	4	M0	M0	M0	M2	M1	M1	M2	M3
7d	3	M1	M1	M3	M3	M0	M0	M0	M2
1d	3	M1	M1	M3	M3	M0	M2	M1	M0
bd	1	M1	M1	M3	M3	M0	M0	M2	M3
e5	2	M1	M1	M3	M1	M1	M3	M3	M3
37	5	M1	M3	M0	M2	M3	M0	M2	M1
25	3	M1	M1	M3	M1	M1	M3	M1	M0
5b	4	M1	M3	M2	M3	M0	M2	M3	M1
a1	5	M1	M1	M1	M0	M0	M2	M1	M2
36	4	M0	M2	M3	M1	M3	M0	M2	M1
70	5	M0	M0	M0	M0	M2	M3	M3	M1
23	4	M1	M3	M2	M1	M1	M3	M1	M0
ae	4	M0	M2	M3	M3	M2	M3	M1	M2
5e	4	M0	M2	M3	M3	M0	M2	M3	M1
3d	2	M1	M1	M3	M3	M0	M0	M2	M1
cd	0	M1	M1	M3	M3	M2	M1	M2	M3
5e	4	M0	M2	M3	M3	M0	M2	M3	M1
8c	4	M0	M0	M2	M3	M2	M1	M0	M2
98	8	M0	M0	M0	M2	M3	M2	M1	M2
ef	3	M1	M3	M0	M0	M2	M3	M3	M3
c7	3	M1	M3	M0	M2	M1	M1	M2	M3
1f	5	M1	M3	M0	M0	M0	M2	M1	M0
61	4	M1	M1	M1	M0	M0	M2	M3	M1
L(M0)		4.62	5.25	4.75	4.40	3.50	3.00	3.50	4.00
L(M1)		3.25	2.56	4.50	3.25	3.50	2.75	4.62	4.00
L(M2)			4.00	4.00	5.00	3.20	4.62	2.71	4.80
L(M3)			4.14	2.73	2.90	5.67	3.71	3.71	2.17
dL(M0)		-1.38	-1.25	-0.75	0.60	-0.30	1.62	-0.79	0.80
dL(M1)			1.59	-1.77	-0.35	2.17	0.96	-0.91	-1.83
d2L(M0)			2.84	-1.02	-0.95	2.47	-0.66	-0.12	-2.63
d2L(M1)			-0.34	2.52	-0.25	-1.87	-2.59	1.70	1.03
(X^Y)[i]		1	1	1	0	1	0	1	0
X[i-1]			1	0	1	1	0	0	1
X[i-1:0]			1	1	5	d	0d	0d	4d
Y[i-1:0]			0	2	2	a	1a	1a	1a

Secret values: X = cd, Y = 9a
Recovered values: X = 4d, Y = 1a
Success

Figure 3: A toy example of successful 8-bit CDPA with 24 traces

2.3.6 A Toy Example

To illustrate the CDPA attack, we provide a toy example with $N = 8$ and with 24 traces Figure 3). (The probability of successfully recovering the two 8-bit secret values using only 24 traces is about 22%; we chose a successful case for this toy example.) The example was produced using the above mentioned Python script in verbose mode, using the following command line:

```
python.exe ./test_cdpa_attack.py -b 8 -t 24 -r 55 -l
```

This toy example is small enough to allow performing all the calculations even manually.

We'll briefly explain the printout in Figure 3. The first 24 rows, excluding the header, correspond to the 24 traces used. In each one of these 24 rows the following data is listed:

- The first column: the pseudo-random value of W (the input data).
- The second column: the value of $HD(H + W, Y)$ (the "trace").
- The remaining columns: the subset M_i to which each trace belongs at each one of steps 0-7

The expressions used for the calculation of the indices i of subsets M_i are written in the header line of Figure 3.

The continuation of the printout is as follows:

- $L(M_i)$ (see (4) and Definition 4) for every step.
- $\partial L(M_i)$ (see (7)) for every step. Only the two rows for $i \in \{0, 1\}$ are shown. The rows for $i \in \{2, 3\}$ are omitted since $\partial L(M_2) = -\partial L(M_0)$ and $\partial L(M_3) = -\partial L(M_1)$. The sign of $\partial L(M_0)$ for step 0 defines the hypothesized value of $X[0] \oplus Y[0]$. To emphasize the sign, this value is printed on a yellow or blue background depending on whether it is negative or positive.
- $\partial^2 L(M_i)$(see (8)) for every step. The rows for $i \in \{2, 3\}$ are omitted for the same reasons as for the case of $\partial L(M_i)$. Which one of values $\partial^2 L(M_0)$ and $\partial^2 L(M_1)$ is greater by absolute value, defines the hypothesized value of $X[i - 1]$, and the sign of this greater value defines the hypothesized value of $X[i] \oplus Y[i]$. To emphasize the sign, the greater value is printed on a yellow or blue background depending on whether it is positive or negative.
- The values of $X[i - 1]$ and $X[i] \oplus Y[i]$ found at each step.
- The cumulative values of $X[i - 1 : 0]$ and $Y[i - 1 : 0]$ found up to each step.

Note that, as explained in the previous sections, the values of $\partial L(M_i)$ approach ± 1, and the values of $\partial^2 L(M_i)$ approach either 0 or ± 2 as the number of traces grows. In this toy example, with only 24 traces, the actual values are still very far from these limits, which could easily cause a failure of the attack, but in this specific case the attack was successful. Figure 4 demonstrates that with a large amount of traces (in this case, 100K) the actual values of $\partial L(M_i)$ and $\partial^2 L(M_i)$ are indeed close to the expected limits. (Only the 8 least significant bits out of 64 are shown.) This printout was produced by the same script as above using the following command line:

```
python.exe ./test_cdpa_attack.py -b 64 -t 100000 -r 3 -v
```

```
         Step 0  Step 1  Step 2  Step 3  Step 4  Step 5  Step 6  Step 7
L(M0)     32.51   32.99   32.02   32.03   33.00   31.99   31.01   31.00
L(M1)     31.50   32.02   31.04   32.98   31.01   32.01   32.03   31.98
L(M2)             32.03   33.00   31.03   31.99   33.01   31.98   32.04
L(M3)             30.97   31.96   31.98   31.99   30.99   32.98   32.98

dL(M0)   -1.01   -0.97    0.98   -0.99   -1.01    1.02    0.97    1.04
dL(M1)            -1.05    0.92   -1.00    0.98   -1.02    0.95    1.00

d2L(M0)           -0.08   -0.06   -0.00    1.99   -2.04   -0.02   -0.05
d2L(M1)            2.02   -1.90    1.99    0.03   -0.01   -1.92   -2.04

(X^Y)[i]     1       1       0       1       1       0       0       0
X[i-1]               0       0       0       1       1       0       0

X[i-1:0]             0       0       0       8      18      18      18
Y[i-1:0]             1       3       3       3      03      03      03

Secret    values: X = 8d01176a121b0698, Y = b5492cf9d706e683
Recovered values: X = 0d01176a121b0698, Y = 35492cf9d706e683
Success
```

Figure 4: An example of successful 64-bit CDPA with 100K traces

3 The Attack on HMAC-SHA-2

3.1 Preliminaries

To describe the attack on HMAC-SHA-2, we start by defining SHA-2 [Nat12] and HMAC [Nat08], using a notation which is more convenient in the context of this paper.

3.1.1 SHA-2

The SHA-2 family of hash algorithms utilizes the Merkle-Dåmgard construction, in which the input (properly padded) is represented as a sequence of blocks $Bl_0, Bl_1, \ldots, Bl_{n-1}$, and the hash function is iteratively calculated as $St_{j+1} = CF(St_j, Bl_j)$ for $0 \leq j < n$ where CF is the *compression function*, St_0 is a predefined constant, and St_n is the final output (hash value).

The compression function $CF(St_j, Bl_j)$ is calculated in the following steps (see SHA-256 example in Figure 5):

1. The message schedule expands the input block Bl_j into a sequence of $s \times N$-bit words $W_0, W_1, \ldots, W_{s-1}$, where $s = 64$, $N = 32$ for SHA-224 and SHA-256, and $s = 80, N = 64$ for SHA-512/224, SHA-512/256, SHA-384 and SHA-512. The details of the expansion algorithm are omitted since they are irrelevant to our attack.

2. The *round function* RF is applied s times so that $R_{r+1} = RF(R_r, W_r, K_r)$ for $r \in [0, 1, \ldots, s-1]$, where K_r are predefined "round constants", and $R_0 = St_j$.

3. Finally, the output of the compression function CF is calculated as a word-wise sum modulo 2^N of $R_0 = St_j$ and R_s.

For the round function RF, the state R_r before round r is split into eight N-bit words. While in [Nat12] these eight words are called A, B, C, D, E, F, G, H, we will denote them

differently — as

$$A_{r-1}, A_{r-2}, A_{r-3}, A_{r-4}, E_{r-1}, E_{r-2}, E_{r-3}, E_{r-4}$$

This notation features the fact that only two words — A_r and E_r — are calculated at round r, while the other words are shifted right in the array (see Figure 6).

A_r and E_r are calculated as follows (where all the addition operations are modulo 2^N):

$$\epsilon_r = E_{r-4} + \Sigma_1(E_{r-1}) + Ch(E_{r-1}, E_{r-2}, E_{r-3}) + K_r \tag{14}$$

$$\alpha_r = \Sigma_0(A_{r-1}) + Maj(A_{r-1}, A_{r-2}, A_{r-3}) \tag{15}$$

$$\Delta E_r = A_{r-4} + \epsilon_r \tag{16}$$

$$\Delta A_r = \epsilon_r + \alpha_r \tag{17}$$

$$E_r = \Delta E_r + W_r \tag{18}$$

$$A_r = \Delta A_r + W_r \tag{19}$$

where Ch is the bit-wise choice function

$$Ch(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z) \tag{20}$$

and Maj is the bit-wise majority function

$$Maj(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) \tag{21}$$

Note that ΔA_r and ΔE_r depend on the previous state R_r but not on W_r. In particular, ΔA_0 and ΔE_0 depend only on the initial state R_0.

This definition is consistent with the standard definition of the round function of any hash function from the SHA-2 family. The difference between different functions in the family is only in N, the constants K_r, and the definitions of the functions Σ_0, Σ_1 (which do not matter for the purpose of our attack).

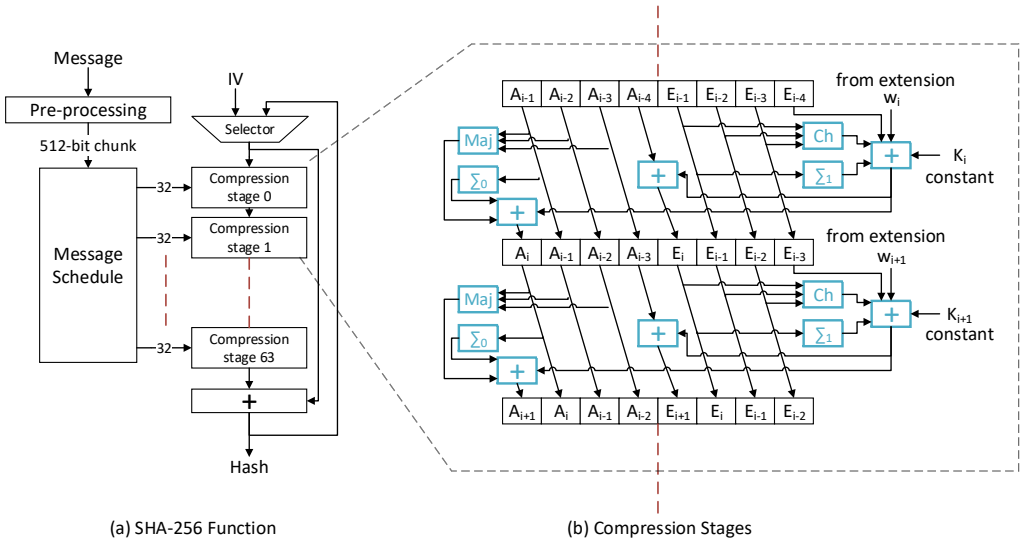

Figure 5: SHA-256 algorithm block diagram. (a) SHA-256 execution flow, including the preprocessing stage, message schedule, which outputs 64×32-bit words, and 64 compression stages. (b) Detailed diagram of two 256-bit wide compression stages.

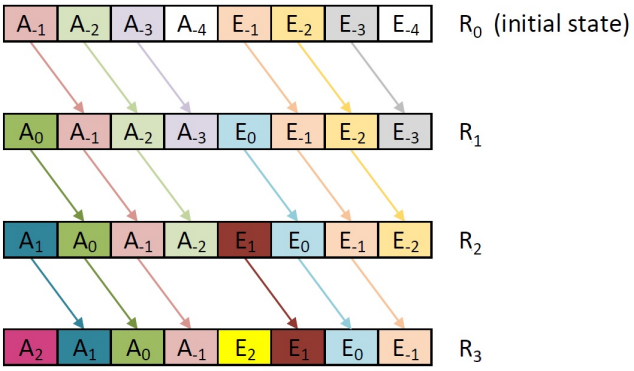

Figure 6: Illustration of the notation used in the paper in the first three rounds. Arrows show copy operations. All the tiles that have incoming edges, receive an exact copy of a word in the previous round. The remaining tiles receive results of manipulated data from the previous round.

3.1.2 HMAC

HMAC is a Message Authentication Code (MAC) algorithm based on a hash function, see (1). The details of the derivation of K_0 from the key K are irrelevant to our article, but it is important to note that, regardless of the size of K, the size of K_0 is equal to the block size of the underlying hash function H. The two applications of the function H during the HMAC calculation are called the "inner" application and the "outer" application.

If H is a function from the SHA-2 family, e.g. SHA-256, then for a fixed K the first application of the SHA-256 compression function in the inner SHA-256 calculates $St_{in} = CF(St_0, K_0 \oplus ipad)$, and the first application of the SHA-256 compression function in the outer SHA-256 calculates $St_{out} = CF(St_0, K_0 \oplus opad)$. Note that both St_{in} and St_{out} depend only on K. The goal of the attack is to find the *secret initial states* St_{in} and St_{out}. Since it is difficult to invert the compression function, it is difficult to derive K or K_0 from St_{in} and St_{out}. However it is not necessary, since an attacker who knows St_{in} and St_{out} can forge $HMAC_{SHA256}(K, M)$ for any message M, which is the ultimate goal of an attack on a MAC algorithm. See Figure 7 which shows HMAC data flow assuming a one-block message including the padding (as there is no reason to use longer messages for the attack).

In order to find St_{in} and St_{out}, both the inner and the outer SHA-256 must be attacked. There is a subtle difference between the two attacks. While the attacker may choose the message M as he wishes in order to attack the inner SHA-256, this is not the situation with the outer SHA-256, because the variable part of its input is the output from the inner SHA-256 — which may become known to the attacker after successfully attacking the inner hash, but in any case cannot be chosen arbitrarily. This makes designing an attack on the outer SHA-256 more difficult. The attack described below works for both, as it does not require choosing the input.

3.2 The Statement of the Problem

We denote the Hamming distance between subsequent states as follows:

$$S_r = HD(R_r, R_{r+1}) \tag{22}$$

for $0 \leq r < n$.

A device calculates the compression function of one of the functions of the SHA-2 family with a secret initial state, *i.e.*, it iteratively invokes the round function with a

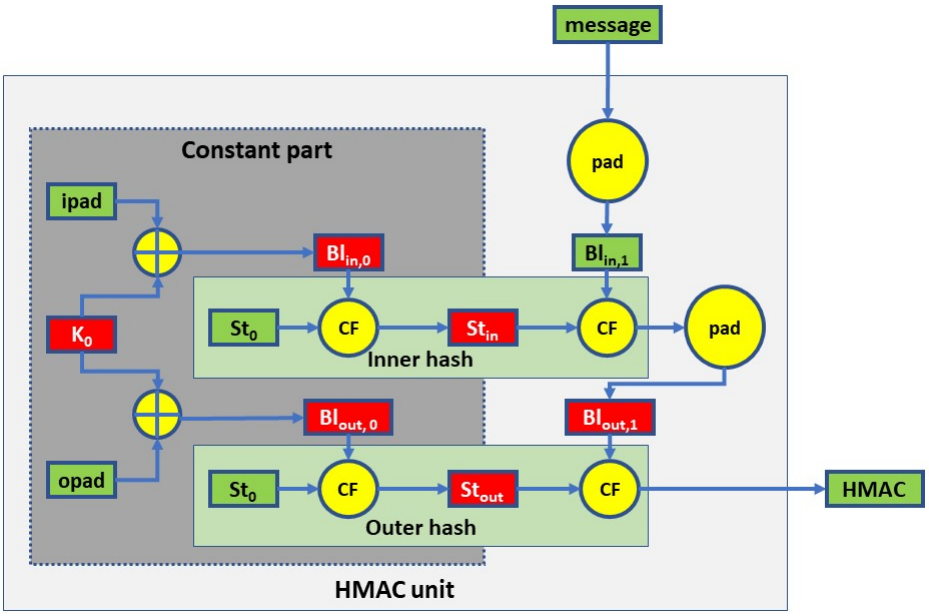

Figure 7

known sequence of input words, starting from the secret initial state R_0. From every such calculation the attacker obtains the sequence of W_r and the corresponding sequence of S_r. The goal is to find R_0.

3.3 Solution Overview

The solution consists of three stages.

At stage 1, all possible information is extracted from the samples corresponding to round 0 (S_0). The result is a list of hypotheses regarding two words of the secret initial state and two words which are functions of the secret initial state.

At stage 2, all possible information is extracted from the samples corresponding to round 1 (S_1). The result is a list of hypotheses regarding the entire secret initial state.

At stage 3, all the incorrect hypotheses are rejected using the samples corresponding to the later rounds, and only the correct hypothesis remains.

The same in more detail.

Stage 1. Analyze round 0. In this round

$$S_0(W_0) = L_{\Delta A_0, A_{-1}}(W_0) + L_{\Delta E_0, E_{-1}}(W_0) + const \tag{23}$$

where *const* is the sum of six Hamming distances corresponding to the replacement of the six constants

$$A_{-2}, A_{-3}, A_{-4}, E_{-2}, E_{-3}, E_{-4}$$

with the six constants

$$A_{-1}, A_{-2}, A_{-3}, E_{-1}, E_{-2}, E_{-3}$$

respectively.

We find $\Delta A_0,\ A_{-1},\ \Delta E_0,\ E_{-1}$ by the analysis of

$$\partial S_0(W_0) = \partial L_{\Delta A_0, A_{-1}}(W_0) + \partial L_{\Delta E_0, E_{-1}}(W_0) \tag{24}$$

which is a modification of the basic CDPA described in Section 2. Two additions instead of one increase the complexity of the task, and the result of stage 1 is a set of 2^k hypotheses regarding ΔA_0, A_{-1}, ΔE_0, E_{-1}, where $k \geq 3$.

Stage 2. Analyze round 1. For every hypothesis from stage 1, either reject it or find all the still unknown words of the secret initial state — A_{-2}, A_{-3}, A_{-4}, E_{-2}, E_{-3}, E_{-4}.

Stage 3. For every hypothesis regarding the full initial internal state it is possible to calculate the Hamming distances at each round. Compare the calculated values against the experimental ones. All the hypotheses except for the correct one will be rejected.

The following sections describe stages 1 and 2.

The attack on SHA2, while it is based on CDPA, is significantly more complex than CDPA. We encourage anyone who wants to understand the attack in detail to try out the Python script at https://github.com/fortify-iq/sha2-attack/blob/master/src/test_sha2_attack.py, in addition to reading the continuation of this paper.

3.4 Stage 1. Find ΔA_0, A_{-1}, ΔE_0, E_{-1}

Stage 1 consists of two substages. At substage 1a we find ΔA_0, ΔE_0, up to the permutation between them, and excluding one most significant bit of each. At substage 1b we find a set of hypotheses for ΔA_0, A_{-1}, ΔE_0, E_{-1}.

3.4.1 Substage 1a. Find ΔA_0, ΔE_0

Similarly to the basic CDPA, we analyze the two simultaneous additions ($\Delta A_0 + W_0$ and $\Delta E_0 + W_0$) in a series of steps. In step i for $1 \leq i < N$ the analysis is modulo 2^{i+1}. Before step i there are j known bits $\Delta A_0[j-1:0]$ and $\Delta E_0[j-1:0]$, where $j < i$. In particular, before step 1, $j = 0$, and no bits are known. The value of j will be discussed below.

Before any step i, there are two cases which we describe separately. We will use the following easy-to-prove propositions.

Proposition 1. *The sum of two odd functions is odd.*

Proposition 2. *The sum of an n_1-step function and an n_2-step function is an (n_1+n_2)-step function.*

Proposition 3. *The sum of an n_1-peak function and an n_2-peak function is an (n_1+n_2)-peak function.*

In particular, since $\partial L_{\Delta A_0, A_{-1}}(W_0)$ and $\partial L_{\Delta E_0, E_{-1}}(W_0)$ are odd 2-step functions, and $\partial^2 L_{\Delta A_0, A_{-1}}(W_0)$ and $\partial^2 L_{\Delta E_0, E_{-1}}(W_0)$ are odd 2-peak functions, from these propositions follows that $\partial S_0(W_0)$ is an odd 4-step function, and $\partial^2 S_0(W_0)$ is an odd 4-peak function.

Case 1. $\Delta A_0[j-1:0] = \Delta E_0[j-1:0]$. Since the j least significant bits are known, modulo 2^{i+1} for both $\partial L_{\Delta A_0, A_{-1}}(W_0)$ and $\partial L_{\Delta E_0, E_{-1}}(W_0)$ there are 2^{i+1-j} intervals at which each function is guaranteed to be constant. Since $i > j$, the number of intervals is at least 4. Since $\Delta A_0[j-1:0] = \Delta E_0[j-1:0]$, these are the same intervals for both functions. Just as in Section 2, we estimate $\partial S_0(M_k)$ for every subset M_k, and find at most two pairs of opposite points at which the odd 4-peak function $\partial^2 S_0(W_0)$ is different from 0. There are three subcases.

Subcase 1a. No non-zero values of $\partial^2 S_0(W_0)$ are found. This means that the peaks of $\partial^2 L_{\Delta A_0, A_{-1}}(W_0)$ and of $\partial^2 L_{\Delta E_0, E_{-1}}(W_0)$ are at the same

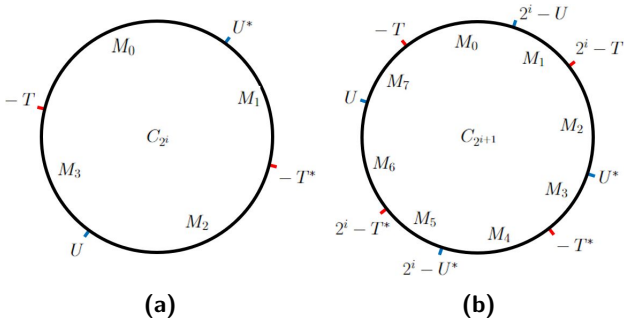

Figure 8

points, but they are opposite by their signs. In this case we gain no information, and proceed to step $i+1$ without changing the number j of the known bits.

Subcase 1b. Two non-zero values of $\partial^2 S_0(W_0)$ are found. This means that the peaks of $\partial^2 L_{\Delta A_0, A_{-1}}(W_0)$ and of $\partial^2 L_{\Delta E_0, E_{-1}}(W_0)$ are at the same points, and the peaks have the same sign, so $\partial^2 S_0(M) = \pm 4$. The pair of points reveals the bits $[i-1:j]$, in addition to the bits $[j-1:0]$ which were already known — and these bit positions still match in the words ΔA_0 and ΔE_0. In this subcase, j (the updated number of known bits) assumes the value i.

Subcase 1c. Four non-zero values of $\partial^2 S_0(W_0)$ are found. This means that the peaks of $\partial^2 L_{\Delta A_0, A_{-1}}(W_0)$ and of $\partial^2 L_{\Delta E_0, E_{-1}}(W_0)$ are at different points, *i.e.*, $\Delta A_0[i-1:0] \neq \Delta E_0[i-1:0]$. In this case from these points we deduce both $\Delta A_0[i-1:0]$ and $\Delta E_0[i-1:0]$ (up to a permutation between them), so $j = i$ bits are known towards the next step $i+1$.

Note that if subcase 1a occurs for several consecutive bit positions, then the number of subsets grows exponentially, and significantly more traces may be necessary for the attack to succeed.

Case 2. $\Delta A_0[j-1:0] \neq \Delta E_0[j-1:0]$. This case occurs for the first time after subcase 1c is encountered, and $j = i-1$ when it happens. After it happens once, clearly this will be the case for all the subsequent bit positions as well. Unlike case 1, in case 2 the number of the known bits before step i is always $j = i-1$, similarly to Section 2. Let's denote $T = \Delta A_0[i-2:0]$ and $U = \Delta E_0[i-2:0]$ (see Figure 8a). Taking into account two options for each one of $\Delta A_0[i-1]$ and $\Delta E_0[i-1]$, modulo 2^{i+1} there are 8 intervals at which $\partial S_0(W_0)$ is guaranteed to be constant as shown in Figure 8b. We estimate the value of $\partial S_0(W_0)$ at each interval, find the two pairs of opposite points at which $\partial^2 S_0(W_0)$ changes its sign, and deduce the bits $\Delta A_0[i-1]$ and $\Delta E_0[i-1]$.

Note that, unlike all the previous cases, the subsets may differ in size. The more consecutive matching bits there are in ΔA_0 and ΔE_0, the more significant the difference in their size is. If this happens, it may significantly increase the number of traces necessary for the attack to succeed.

3.4.2 Substage 1b. Find A_{-1}, E_{-1}

This substage consists of a series of steps numbered from 1 to $N-1$.

In step 1 we are going to find $A_{-1}[1:0],\ E_{-1}[1:0]$. The analysis will be modulo 4. We split the traces into four subsets M_k $(0 \le k < 4)$, according to $W_0[1:0]$, estimate $S_0(M_k)$, and calculate

$$\Delta S_0(k) = S_0(M_{k+1}) - S_0(M_k) \tag{25}$$

for $0 \le k < 3$. On the other hand, for every one of the sixteen possible combinations of bits $A_{-1}[1:0], E_{-1}[1:0]$ we calculate the expected values of $\Delta S_0(k)$ for $0 \le k < 3$, using the explicit expression of $HD_{[1:0]}(\Delta A_0 + W_0, A_{-1}) + HD_{[1:0]}(\Delta E_0 + W_0, E_{-1})$. As a result, most of the combinations are rejected. The set of the remaining combinations is the set of hypotheses for the next step.

As long as $\Delta A_0[i-2:0] = \Delta E_0[i-2:0]$, step i for every hypothesis is similar to step 1, with the following differences:

1. The analysis is modulo 2^{i+1}.
2. The bits $A_{-1}[i-1:0]$ and $E_{-1}[i-1:0]$ are known from the previous steps.
3. The target bits are $A_{-1}[i:i-1],\ E_{-1}[i:i-1]$.
4. The splitting into four subsets is according to $(\Delta A_0[i-2:0] + W)[i:i-1]$.
5. In addition to rejecting the combinations of $A_{-1}[i:i-1],\ E_{-1}[i:i-1]$ because of the mismatch between the measured and expected values, they may be rejected because of the mismatch with the already known values of bits $A_0[i-1]$ and $E_0[i-1]$. (It still may happen that more than one combination will remain.)

As soon as $\Delta A_0[i-2:0] \ne \Delta E_0[i-2:0]$, everything becomes simpler because it is possible to separate between ΔA_0 and ΔE_0. The bits $\Delta A_0[N-2:0]$ and $\Delta E_0[N-2:0]$ are known from substage 1, and the analysis is modulo 2^{i+1}. We split $C_{2^{i+1}}$ into 8 subsets in the same way as in Section 3.4.1, case 2 (see Figure 8b), and deduce the only possible values of $\Delta A_0[i] \oplus A_{-1}[i]$ and of $\Delta E_0[i] \oplus E_{-1}[i]$ from the sign of $\partial S_0(W_0)$ in the points $\Delta A[i-1:0]$ and $\Delta E[i-1:0]$, according to (12). At all the steps except for the step $N-1$ (the last one), $\Delta A_0[i]$ and $\Delta E_0[i]$ are already known, so $A_{-1}[i]$ and $E_{-1}[i]$ are easily found. At the last step $i = N-1$, and we remain with $\Delta A_0[N-1] \oplus A_{-1}[N-1]$ and $\Delta E_0[N-1] \oplus E_{-1}[N-1]$.

3.4.3 Wrapping Up Stage 1

After all the steps described above, we have a list of hypotheses regarding

$$\langle A_0[N-2:0],\ \Delta A_0[N-2:0],\ \Delta A_0[N-1] \oplus A_{-1}[N-1],\\ E_0[N-2:0],\ \Delta E_0[N-2:0],\ \Delta E_0[N-1] \oplus E_{-1}[N-1] \rangle$$

We convert each hypothesis into four hypotheses regarding

$$\langle A_0[N-1:0],\ \Delta A_0[N-1:0],\ E_0[N-1:0],\ \Delta E_0[N-1:0] \rangle$$

by listing all the combinations of $A_{-1}[N-1]$ and $E_{-1}[N-1]$. Note that there is a total of at least eight hypotheses, because of the permutations between the pairs ΔA_0, A_{-1} and ΔE_0, E_{-1}.

3.5 Stage 2. Find the Rest of the Secret Initial Stage

After stage 1 we have a set of hypotheses regarding

$$\langle A_0[N-1:0],\ \Delta A_0[N-1:0],\ E_0[N-1:0],\ \Delta E_0[N-1:0] \rangle$$

At stage 2 we analyze each one of these hypotheses separately. As a result, each hypothesis is either rejected or expanded into a hypothesis regarding the entire initial internal state.

For the purpose of this stage, we'll denote

$$AE_{-3} = A_{-3} + E_{-3} \tag{26}$$

The analysis is performed in steps numbered from 0 to $N-1$. In step i we find $A_{-2}[i]$, $AE_{-3}[i]$, $E_{-2}[i]$, $E_{-3}[i]$. If all the steps succeed, we wrap up by a simple calculation of the remaining words — A_{-3}, A_{-4}, E_{-4}.

The analysis is based on the following observations.

1. After stage 1, A_{-1} and E_{-1} are known.
2. After stage 1, A_0, $\Sigma_0(A_0), E_0, \Sigma_1(E_0)$ are known for every W_0.
3. If M_0 and M_1 are two subsets of the traces chosen according to criteria related to the calculation of $A_1[i]$ (or $E_1[i]$), then in the expression for $S_1(M_0) - S_1(M_1)$ all the terms except $HD(A_1[i], A_0[i]) = A_1[i] \oplus A_0[i]$ (or $HD(E_1[i], E_0[i]) = E_1[i] \oplus E_0[i]$) are distributed uniformly in both sets and therefore almost cancel out for sufficiently large subsets M_0 and M_1.
4. If $E_{-2}[i-1:0]$ and $AE_{-3}[i-1:0]$ are known, then in the expression for $E_1[i] \oplus E_0[i]$ the only unknown values are $ch(E_0[i], E_{-1}[i], E_{-2}[i])$ and $AE_{-3}[i]$.
5. If $E_{-2}[i:0]$, $AE_{-3}[i:0]$, $A_{-2}[i-1:0]$ and $A_{-3}[i-1:0]$ are known, then in the expression for $A_1[i] \oplus A_0[i]$ the only unknown values are $maj(A_0[i], A_{-1}[i], A_{-2}[i])$ and $E_{-3}[i]$.

Taking these observations into account, step i is performed as follows.

1. Split the traces into two subsets M_0 and M_1 according to the bit $E_0[i]$.
2. Note that if $E_0[i] = 1$, then according to (20)

   $$ch(E_0[i], E_{-1}[i], E_{-2}[i]) = E_{-1}[i]$$

   which is known, and the only remaining unknown term in the expression for $E_1[i] \oplus E_0[i]$ is $AE_{-3}[i]$, *i.e.*,

   $$E_1[i] \oplus E_0[i] = AE_{-3}[i] \oplus Q$$

   where Q is known. Split M_1 into two subsets M_{10} and M_{11} according to the value of Q. From the sign of $S_1(M_{11}) - S_1(M_{10}) \approx \pm 1$ deduce the value of $AE_{-3}[i]$.
3. Note that if $E_0[i] = 0$, then according to (20)

   $$ch(E_0[i], E_{-1}[i], E_{-2}[i]) = E_{-2}[i]$$

   which is now the only remaining unknown term in the expression for $E_1[i] \oplus E_0[i]$, *i.e.*,

   $$E_1[i] \oplus E_0[i] = E_{-2}[i] \oplus Q$$

   where Q is known. Split M_0 into two subsets M_{00} and M_{01} according to the value of Q. From the sign of $S_1(M_{01}) - S_1(M_{00}) \approx \pm 1$ deduce the value of $E_{-2}[i]$.
4. Split the traces into two subsets M_0 and M_1 according to $A_0[i] \oplus A_{-1}[i]$.

5. Note that if $A_0[i] = A_{-1}[i]$ (*i.e.*, $A_0[i] \oplus A_{-1}[i] = 0$), then according to (21)

$$maj(A_0[i], A_{-1}[i], A_{-2}[i]) = A_{-1}[i]$$

which is known, and the only remaining unknown term in the expression for $A_1[i] \oplus A_0[i]$ is $E_{-3}[i]$, *i.e.*,

$$A_1[i] \oplus A_0[i] = E_{-3}[i] \oplus Q$$

where Q is known. Split M_0 into two subsets M_{00} and M_{01} according to the value of Q. From the sign of $S_1(M_{01}) - S_1(M_{00}) \approx \pm 1$ deduce the value of $E_{-3}[i]$.

6. Note that if $A_0[i] \neq A_{-1}[i]$ (*i.e.*, $A_0[i] \oplus A_{-1}[i] = 1$), then according to (21)

$$maj(A_0[i], A_{-1}[i], A_{-2}[i]) = A_{-2}[i]$$

which is now the only remaining unknown term in the expression for $A_1[i] \oplus A_0[i]$, *i.e.*,

$$A_1[i] \oplus A_0[i] = A_{-2}[i] \oplus Q$$

where Q is known. Split M_1 into two subsets M_{10} and M_{11} according to the value of Q. From the sign of $S_1(M_{11}) - S_1(M_{10}) \approx \pm 1$ deduce the value of $A_{-2}[i]$.

If at any step, one of the values expected to be close to ± 1 is close to 0 instead, the hypothesis is rejected.

3.5.1 Wrapping Up Stage 2

Once all the steps are finished, we calculate

$$A_{-3} = AE_{-3} - E_{-3}$$

and calculate A_{-4} and E_{-4} based on the already known values of

$$\Delta A_0,\ A_{-1},\ A_{-2},\ A_{-3},\ \Delta E_0,\ E_{-1},\ E_{-2},\ E_{-3}$$

3.6 The Case of Two Rounds per Clock Cycle Implementation

If two rounds are calculated in one clock cycle, the attack still works with minor changes.

In this case, at clock cycle 0 two rounds — round 0 and round 1 — are calculated, and the Hamming distance obtained at this clock cycle is $S_0^* = HD(R_0, R_2)$. It includes four non-constant addends

$$HD(A_0, A_{-2}) + HD(E_0, E_{-2}) + HD(A_1, A_{-1}) + HD(E_1, E_{-1}) \tag{27}$$

Performing stage 1 of the attack described in Section 3.4, with S_0^* instead of S_0, produces a set of hypotheses regarding $\Delta A_0,\ A_{-2},\ \Delta E_0,\ E_{-2}$. (The addends corresponding to the terms $HD(A_1, A_{-1})$ and $HD(E_1, E_{-1})$ almost cancel out, as the criteria of splitting into subsets are irrelevant to them.)

For stage 2 we also use S_0^* (instead of S_1). This time, the addends corresponding to $HD(A_0, A_{-2})$ and $HD(E_0, E_{-2})$ almost cancel out for similar reasons. A_{-1} and E_{-1} are not known yet — but on the other hand A_{-2} and E_{-2} are known. The order of performing step i of stage 2 in this case is as follows.

1. Split the traces into two subsets M_0 and M_1 according to the bit $E_0[i]$.

2. Note that if $E_0[i] = 0$, then according to (20)

$$ch(E_0[i], E_{-1}[i], E_{-2}[i]) = E_{-2}[i]$$

which is known, and the only remaining unknown term in the expression for $E_1[i] \oplus E_{-1}[i]$ is $AE_{-3}[i]$, *i.e.*,

$$E_1[i] \oplus E_{-1}[i] = AE_{-3}[i] \oplus Q$$

where Q is known. Split M_0 into two subsets M_{00} and M_{01} according to the value of Q. From the sign of $S_0^*(M_{01}) - S_0^*(M_{00}) \approx \pm 1$ deduce the value of $AE_{-3}[i]$.

3. Note that if $E_0[i] = 1$, then according to (20)

$$ch(E_0[i], E_{-1}[i], E_{-2}[i]) = E_{-1}[i]$$

which is now the only remaining unknown term in the expression for $E_1[i] \oplus E_{-1}[i]$, *i.e.*,

$$E_1[i] \oplus E_{-1}[i] = E_{-1}[i] \oplus Q$$

where Q is known. Split M_1 into two subsets M_{10} and M_{11} according to the value of Q. From the sign of $S_0^*(M_{11}) - S_0^*(M_{10}) \approx \pm 1$ deduce the value of $E_{-1}[i]$.

4. Split the traces into two subsets M_0 and M_1 according to $A_0[i] \oplus A_{-2}[i]$.

5. Note that if $A_0[i] = A_{-2}[i]$ (*i.e.*, $A_0[i] \oplus A_{-2}[i] = 0$), then according to (21)

$$maj(A_0[i], A_{-1}[i], A_{-2}[i]) = A_{-2}[i]$$

which is known, and the only remaining unknown term in the expression for $A_1[i] \oplus A_{-1}[i]$ is $E_{-3}[i]$, *i.e.*,

$$A_1[i] \oplus A_{-1}[i] = E_{-3}[i] \oplus Q$$

where Q is known. Split M_0 into two subsets M_{00} and M_{01} according to the value of Q. From the sign of $S_0^*(M_{01}) - S_0^*(M_{00}) \approx \pm 1$ deduce the value of $E_{-3}[i]$.

6. Note that if $A_0[i] \neq A_{-2}[i]$ (*i.e.*, $A_0[i] \oplus A_{-2}[i] = 1$), then according to (21)

$$maj(A_0[i], A_{-1}[i], A_{-2}[i]) = A_{-1}[i]$$

which is now the only remaining unknown term in the expression for $A_1[i] \oplus A_{-1}[i]$, *i.e.*,

$$A_1[i] \oplus A_{-1}[i] = A_{-1}[i] \oplus Q$$

where Q is known. Split M_1 into two subsets M_{10} and M_{11} according to the value of Q. From the sign of $S_0^*(M_{11}) - S_0^*(M_{10}) \approx \pm 1$ deduce the value of $A_{-1}[i]$.

4 The Real World Setting and the Heuristics Used in it

4.1 The Challenge of Applying CDPA to HMAC-SHA-2

The attack described in Section 3 is based on the assumption of the Hamming distance leakage model. In this assumption, if (theoretically) all pairs $\langle W_0, W_1 \rangle$ have been used in the experiments, then the equality (13) and other similar equalities are exact, and the attack definitely works. If only a subset of possible values of $\langle W_0, W_1 \rangle$ is used (which in practice is always the case), then the equalities are approximate — but they approach the theoretical limit as the number of the experiments grows (assuming, of course, that the bits of $\langle W_0, W_1 \rangle$ are distributed independently and uniformly). In real world devices,

however, inevitably there is noise added to the Hamming distance. This noise can be of two kinds — uncorrelated and correlated.

The uncorrelated noise (*e.g.*, the thermal noise, and power consumption of unrelated parts of the device) raises the number of traces necessary for revealing the secret, but eventually approaches zero and does not preclude the attack. However the correlated noise may totally thwart the attack, since it may change not only the rate at which the experimental results approach the theoretical limit, but also change the limit which they approach and make it differ from the theoretical limit.

All the theoretical limits in the attack are integers, so sufficiently small deviations of the actual limit from the theoretical one are tolerable. However, if the deviation is too large, then the attack will not work, or will require some changes in order to work.

Indeed, when performing the attack on an FPGA board (see full details in Section 5), we observed this deviation of the limit from the theoretical value.

We believe that measuring EM radiation in the vicinity of the registers, rather than power consumption, would decrease correlated noise from the combinational logic and thus improve the signal-to-noise ratio and diminish the observed deviation from the limit. However we did not verify this assumption; this is a subject for future research.

4.2 The Heuristics Used for the Attack on an FPGA Board

In order to raise our chances of success, we enhanced the algorithm with several heuristics, namely:

1. At stage 1, instead of using a single sample, we take into account several samples (from the same clock cycle). Unlike the experiments in simulation, with just one sample per round, in the experiments on an FPGA board we have several samples per round (4 samples in our case), and we do not know in advance which of them are best correlated with the Hamming distance.

2. Moreover, we take samples from several clock cycles, *i.e.*, from several rounds. (Note that $HD(A_0, A_{-1})$ is an addend in the expression for S_r for $0 \le r < 4$. For this reason, taking samples from up to 4 consecutive clock cycles may be advantageous.)

3. We normalize the samples used (where the normalization parameters are per sample). (The values of $\partial^2 S_0(W_0)$ in Section 3.4.1, of $\Delta S_0(k)$ in Section 3.4.2, and of several differences between averages in Section 3.5 are each expected to approach some small integer as the number of traces approaches infinity, as explained in the above mentioned sections. However this is true only on the assumption that the samples used are equal to the Hamming distance between consecutive states, optionally with some added noise. In the real world, the best we can hope for is a linear dependency between the sample and the Hamming distance, with the coefficient not necessarily 1 and the free term not necessarily 0. In order to make the comparisons with the expected small integers meaningful, we need to normalize the samples. For this purpose, we perform measurements with a known initial internal state, and find the coefficient (per sample), multiplication by which causes the above mentioned values to be on average as close to the expected integers as possible. Unlike all our other heuristic changes, this one would be necessary even if there were no noise.)

4. At every step, we perform the calculation (*e.g.*, the values of $\partial^2 S_0(W_0)$) separately for every sample, and then average over the samples.

5. In case 1 of stage 1 (where the least significant bits of ΔA_0 and ΔE_0 match), we start from step 2 rather than 1. Moreover, each time when subcase 1b occurs, and the number of known bits j assumes the value i (the step counter), we proceed directly to step $i+2$ skipping step $i+1$, so that always $j < i-1$ rather than $j < i$,

i.e., there are always at least 8 rather than at least 4 subsets. The goal is to make the distinction clearer between subcases 1a and 1c (no non-zero values of $\partial^2 S_0(W_0)$ found vs. four non-zero values of $\partial^2 S_0(W_0)$ found).

6. In general, at every step we find some bit, or bits, depending on the matching between the experimental average values and one of the set of the possible theoretical values. At stage 1, if at any step the differences between the experimental result and all theoretical options are close enough to one of the theoretical values, *i.e.*, the difference is less than a certain threshold, we regard it as a match. If all the differences are greater than this threshold, we take all the options into account. If after several steps the number of options exceeds (another) threshold, the options with the highest score are dropped, where the score is calculated as a sum of the deviations from the theoretical values at all the steps.

7. At stage 3, instead of simply comparing the measured leakage against the expected Hamming distance (which would not work due to noise), we calculate the correlation between the measured leakage and the expected Hamming distance at several first rounds. While for the incorrect hypotheses the correlation rapidly decreases towards the noise level, for the correct hypothesis the correlation remains significant.

5 Experimental Results

In order to assess the properties of the attack, we mounted it in two settings:

1. Software simulation in the Hamming distance model of leakage.

2. FPGA board. Here we added the "real world" heuristics described in Section 4.2.

In the next sections we describe the methodology and the results of both types of experiments. The target hash function was SHA-256 in all the experiments.

5.1 Experiments in Simulation

Table 2: The number of traces for success probability $> 50\%$

Noise	SHA256	SHA512
0	2^{16}	2^{18}
4	2^{17}	2^{19}
8	2^{18}	2^{20}
16	2^{20}	$> 2^{20}$
32	$> 2^{20}$	

We simulated in software the Hamming distance model without noise (neither correlated nor uncorrelated). Using this software simulation to generate traces, we performed experiments with different amounts of traces — from 10K to 50K with a 5K step, and from 50K to 1M with a 50K step. For each amount of traces, we performed 1000 experiments with different randomly chosen secret initial states. For stage 1 (Section 3.4) we used the entire amount of traces each time. Whenever the amount of traces was greater than 20K, for stage 2 (Section 3.5) we used only the first 20K traces, since even in this setting attack failures at this stage were extremely rare. The results are shown in Figure 9. The probability of success is 4.6% for 15K traces and rapidly grows, reaching 90% at 200K traces and 99% at 1M traces.

In addition, we performed a similar analysis for a single key — the same key as was used on the FPGA board. For this single key the success rate starts from 4.1% with 15K traces, and reaches $\approx 100\%$ with 130K (see Figure 10).

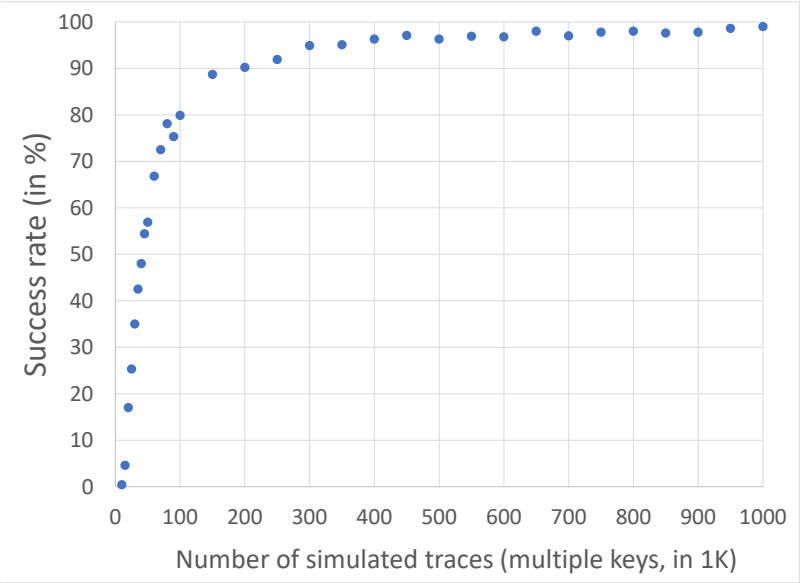

Figure 9: The success rate in % as a function of the number of simulated traces with multiple SHA256 initial states

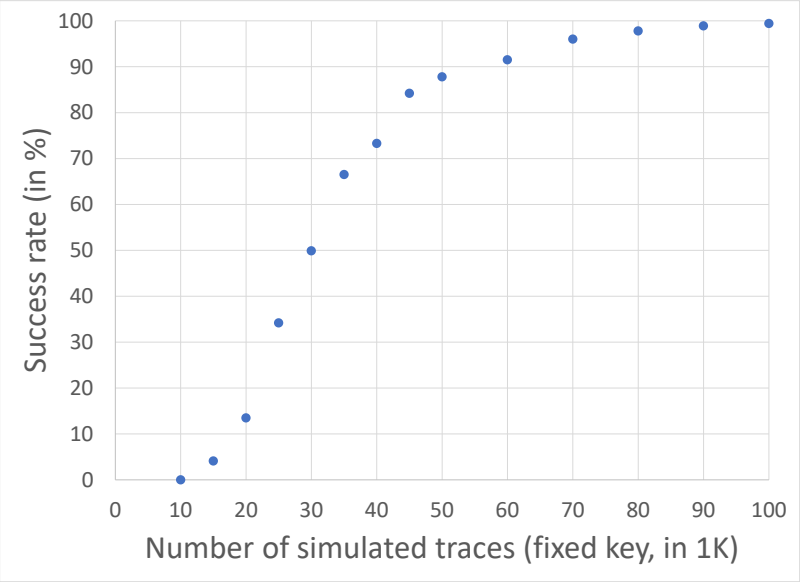

Figure 10: The success rate in % as a function of the number of simulated traces with a fixed SHA256 initial state

In order to study how noise affects the results, and to make conclusions regarding high order attacks on SHA2, we used a Python script publicly available at https://github.com/fortify-iq/sha2-attack/blob/master/src/test_sha2_attack.py. This script generates simulated traces of the two first rounds of the compression function of SHA256 or SHA512 with a secret initial state. Using this script, we collected statistical data for success metrics, similarly to what we did for CDPA (Section 2.3.5), with the following differences:

- We did not use the percentage of correctly found bits as a success metric, because in the case of failure, our implementation of the attack on SHA2 typically raises an exception and finishes prematurely — unlike CDPA which always finds all the bits, either correctly or incorrectly.
- We performed only first order analysis, since even second order attacks on SHA2 are very difficult and hardly practical, as explained below.

The full results of this study are presented in the Excel file https://github.com/fortify-iq/sha2-attack/blob/master/docs/sha2_attack_stats.xlsx. A short summary of the results is presented in Table 2. Here also, similarly to CDPA, the metrics are highly correlated, and in the same way as for CDPA, so taking the condition $M_2 > 65\%$ instead of $M_1 > 50\%$ (in the notation of Section 2.3.5) would not change the table at all.

Comparing Table 1 and Table 2, as well as Figure 11 and Figure 12, we observe that a (first order) attack on SHA256 requires about two orders of magnitude more traces than a 32-bit CDPA. CDPA second order attacks require two orders of magnitude more traces than first order attacks. Attacking SHA2 is much more complicated, and we assume that the ratio between the second and first order attacks on SHA2 is at least the same as for CDPA, *i.e.*, two orders of magnitude. The minimal number of traces for a non-negligible probability of a successful first order attack on SHA256 (32 bit) with zero noise, is approximately 10K traces. By extrapolation, for a second order attack with zero noise non-negligible probabilities start not before 1M traces, which means that the attack, although theoretically possible, is very difficult, and third order attacks are totally impractical. Note that, as was already mentioned (Section 3.4.1, notes to both cases 1 and 2), matching bit sequences in ΔA_0 and ΔE_0 significantly increase the number of traces required for the success of the attack unlike CDPA, for which the complexity of the attack does not depend on the secret values. This is the reason why the slopes in Figure 11 (attack on SHA256) are much less steep than in Figure 12 (32-bit first order CDPA), and even 1M traces do not suffice in approximately 1% of the attacks on SHA256.

5.2 Experiments on an FPGA Board

5.2.1 Setup

To evaluate the proposed method we took a low-area SHA-256 realization from [Str]. We synthesized the RTL for a `CW305 Artix` FPGA target board by NewAE Technology [OC14] with the `Keysight E36100B Series` DC Power Supply for power stabilization. The traces were collected using the NewAE Technology `ChipWhisperer-Lite` kit with four samples per clock cycle. The power signal was obtained by measuring current via a shunt resistor connected serially to the FPGA supply line.

5.2.2 Methodology

On the FPGA board we used the following methodology.

1. Generate 1M traces for a single secret initial state and bit-wise uniformly and independently distributed input data.

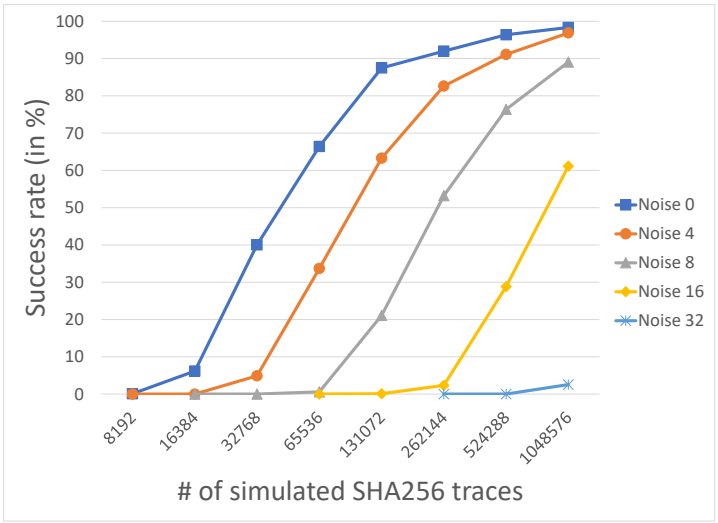

Figure 11: The success rate in % as a function of the number of traces and the noise amplitude for the attack on SHA256

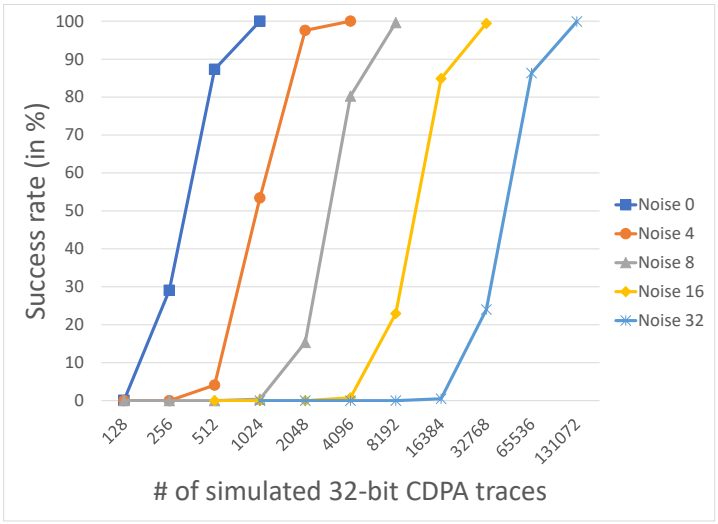

Figure 12: The success rate in % as a function of the number of traces and the noise amplitude for the 32-bit first order CDPA

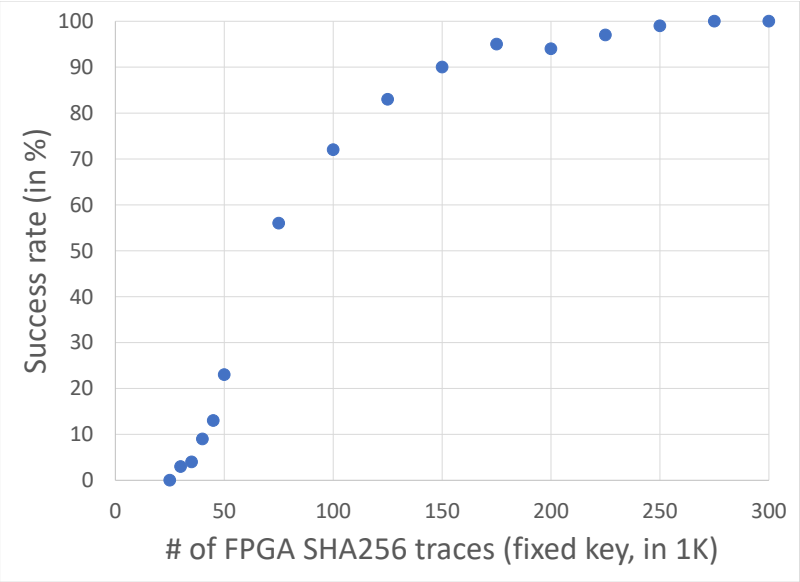

Figure 13: The success rate in % as a function of the number of traces measured on the FPGA board with a fixed SHA256 initial state

2. Pick 100 random subsets of a fixed size.
3. Perform the attack based on each of the subsets.
4. Calculate the success rate.
5. Repeat steps 2–4 for subset sizes from 25K to 50K with a 5K step, and from 50K to 300K with a 25K step.

The results are shown in Figure 13. With 30K traces we already observed a 3% success rate, which is significantly better than in the previous full-fledged attack on HMAC-SHA-2 in pure parallel hardware[BDT+21], not to mention dropping the requirement of the profiling stage. With 275K and 300K traces, 100 out of 100 attack attempts were successful.

5.2.3 Time and Disk Space.

The trace acquisition ran at the rate of 11.5K traces per minute, so in order to acquire 300K traces 26 min sufficed. The disk space necessary for 300K traces, with 288 four-byte samples per trace (one application of the SHA-256 compression function), is about 350 MB. For full HMAC-SHA-256 traces (two applications of the SHA-256 compression function in the inner hash and two applications in the outer hash) 4 times more space will be required. On the other hand, only a few rounds of two applications are used for the attack, so it is easy to drop most of the data by preprocessing. Even without such preprocessing, these requirements for the disk space are not burdensome.

Stage 1 took about 16 seconds, resulting (for the key we worked with) in 320 hypotheses to be checked at stage 2. Handling the correct hypothesis at stage 2 took about 830 ms. Incorrect hypotheses take less time, because they are eventually rejected before performing all the steps, so the total time for stage 2 is less than 4 min.

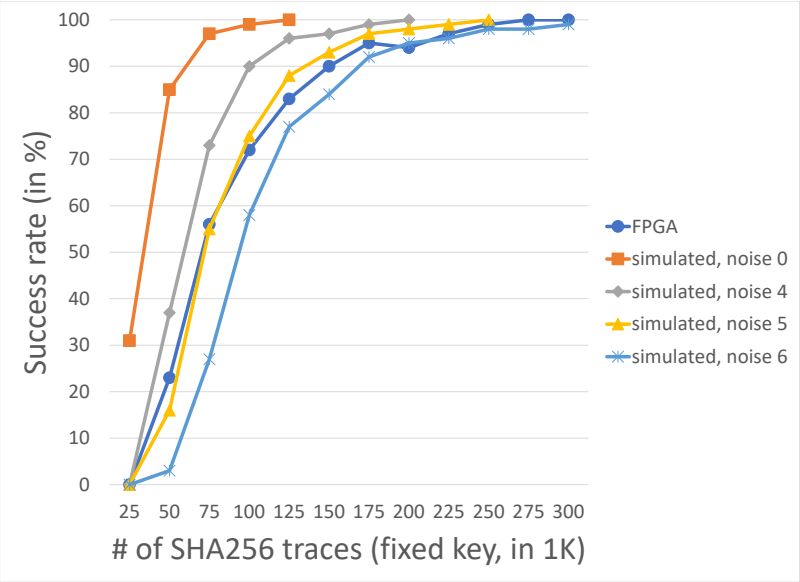

Figure 14: The success rate in % as a function of the number of traces for FPGA traces and for simulated traces with added noise

Summarizing, for the key we worked with, the complete attack on 300K traces, including trace acquisition and analysis, took about half an hour.

5.2.4 Estimation of the Noise in the FPGA Simulation

In order to estimate the level of noise in our FPGA measurements, we performed experiments in simulation with the same key as in the experiments on the FPGA board, with a varying number of traces, and with different levels of added noise. As shown in Figure 14, the dependency of the success rate on the number of traces for the FPGA measurements is approximately the same as such a dependency for simulated traces with added noise at amplitude 5.

6 Conclusions

In this article, we presented a novel DPA attack methodology on arithmetic addition — the Carry-based DPA (CDPA). We demonstrated a full-fledged attack on HMAC-SHA-2 in pure parallel hardware, assuming the Hamming distance leakage model and the performance of either one or two rounds per clock cycle. We validated the methodology both in software simulation and on an FPGA board. The experimental data from the FPGA board showed that there exist attack settings in which the secrets can be revealed, where as few as 30K traces may be sufficient for a successful attack (in 3% of the cases). This is a significant improvement over the best published full attack on HMAC-SHA-2 in hardware [BDT+21]. Moreover, our attack does not require profiling. It is true even for power consumption measurements which we performed, and we expect that EM radiation measurements give even better chances to the attacker. Therefore, implementations of HMAC-SHA-2 in pure

parallel hardware are vulnerable to side-channel attacks, unless they are properly defended.

Additionally, we demonstrated that CDPA can be applied to implementations which represent data in shares (high order CDPA), but the complexity of the attack grows rapidly, and the third order attack is already impractical. While the same technique of using higher moments instead of averages, which we used in high order CDPA, can be applied to the attack on HMAC-SHA-2 as well, we estimate that this attack is very difficult, if at all practically possible, for the second order, and totally impractical for the third order.

Future research may concentrate on further improving the results of this work by measuring EM radiation with a properly placed probe to reduce the impact of the correlated noise, and on designing CDPA-based attacks on other algorithms which use arithmetical addition, *e.g.*, ARX ciphers.

References

[APSQ06] C. Archambeau, E. Peeters, F. X. Standaert, and J. J. Quisquater. Template attacks in principal subspaces. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 4249 LNCS, pages 1–14. Springer Verlag, 10 2006.

[BBD+15] Sonia Belaíd, Luk Bettale, Emmanuelle Dottax, Laurie Genelle, and Franck Rondepierre. Differential power analysis of HMAC SHA-1 and HMAC SHA-2 in the hamming weight model. In *Communications in Computer and Information Science*, volume 554, pages 363–379. Springer Verlag, 2015.

[BDT+21] Yaacov Belenky, Ira Dushar, Valery Teper, Hennadii Chernyshchyk, Leonid Azriel, and Yury Kreimer. *First Full-Fledged Side Channel Attack on HMAC-SHA-2*, volume 12910 LNCS. Springer International Publishing, 2021.

[BSI13] BSI. Anwendungshinweise und Interpretationen zum Schema (AIS) 46. Technical report, BSI, 2013.

[FLRV09] Pierre-Alain Fouque, Gaëtan Leurent, Denis Réal, and Frédéric Valette. Practical Electromagnetic Template Attack on HMAC. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 5747 LNCS, pages 66–80. Springer, 2009.

[GWM16] Catherine H. Gebotys, Brian A. White, and Edgar Mateos. Preaveraging and carry propagate approaches to side-channel analysis of HMAC-SHA256. *ACM Transactions on Embedded Computing Systems*, 15(1):1–19, 2 2016.

[KGB+18] Matthias J. Kannwischer, Aymeric Genêt, Denis Butin, Juliane Krämer, and Johannes Buchmann. Differential power analysis of XMSS and SPHINCS. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10815 LNCS:168–188, 2018.

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In *Annual international cryptology conference*, pages 388–397. Springer, Berlin, Heidelberg, 1999.

[Koc96] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, RSA, DSS, and other systems. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 1109, pages 104–113. Springer Verlag, 1996.

[MTMM07] Robert McEvoy, Michael Tunstall, Colin C. Murphy, and William P. Marnane. Differential power analysis of HMAC based on SHA-2, and countermeasures. In *Cryptographic Hardware and Embedded Systems – CHES 2007*, volume 4867 LNCS, pages 317–332. Springer Verlag, 2007.

[Nat08] National Institute of Standards and Technology. FIPS 198-1: The Keyed-Hash Message Authentication Code, 2008.

[Nat12] National Institute of Standards and Technology. FIPS 180-4: Secure Hash Standard (SHS), 2012.

[Nat15] National Institute of Standards and Technology. FIPS 202: SHA-3 Standard : Permutation-Based Hash and Extendable-Output Functions, 2015.

[OC14] Colin O'flynn and Zhizhang Chen. ChipWhisperer: An open-source platform for hardware embedded security research. In Emmanuel Prouff, editor, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 8622 LNCS, pages 243–260, Cham, 2014. Springer International Publishing.

[Osw16] David Oswald. Side-channel attacks on SHA-1-based product authentication ICs. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 9514, pages 3–14. Springer Verlag, 2016.

[RM13] Pankaj Rohatgi and Mark Marson. NSA Suite B Crypto, Keys, and Side Channel Attacks, 2013.

[Str] Joachim Strömbergson. secworks/sha256: Hardware implementation of the SHA-256 cryptographic hash function.