Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

Correlation power attack on a message authentication code based on SM3*

Ye YUAN[†1,2], Kai-ge QU[†‡1,2], Li-ji WU[†‡1,2], Jia-wei MA[3], Xiang-min ZHANG[†1,2]

[1]*Institute of Microelectronics, Tsinghua University, Beijing 100084, China*

[2]*National Laboratory for Information Science and Technology, Tsinghua University, Beijing 100084, China*

[3]*State Key Laboratory of Cryptography, Beijing 100094, China*

[†]E-mail: yuan-y15@mails.tsinghua.edu.cn; kaigequ@gmail.com; lijiwu@tsinghua.edu.cn; zhxm@mail.tsinghua.edu.cn

Received May 19, 2018; Revision accepted July 30, 2018; Crosschecked July 12, 2019

Abstract: Hash-based message authentication code (HMAC) is widely used in authentication and message integrity. As a Chinese hash algorithm, the SM3 algorithm is gradually winning domestic market value in China. The side channel security of HMAC based on SM3 (HMAC-SM3) is still to be evaluated, especially in hardware implementation, where only intermediate values stored in registers have apparent Hamming distance leakage. In addition, the algorithm structure of SM3 determines the difficulty in HMAC-SM3 side channel analysis. In this paper, a skillful bit-wise chosen-plaintext correlation power attack procedure is proposed for HMAC-SM3 hardware implementation. Real attack experiments on a field programmable gate array (FPGA) board have been performed. Experimental results show that we can recover the key from the hypothesis space of 2^{256} based on the proposed procedure.

Key words: HMAC-SM3; Side channel analysis; Correlation power attack; Bit-wise chosen-plaintext
https://doi.org/10.1631/FITEE.1800312 **CLC number:** TP309

1 Introduction

Message authentication code (MAC) (Menezes et al., 1996) provides message integrity in many security applications. Hash-based message authentication code (HMAC) (Bellare et al., 1996; FIPS, 2002) is a popular type of MAC, which is constructed by a hash algorithm and a key. SM3 (SCA, 2010; Guo et al., 2015) is a hash algorithm proposed by the National Cryptography Administration of China as a national standard. HMAC-SM3 side channel security implementation remains to be studied further.

Typically, differential power analysis (DPA) (Kocher et al., 1999) and correlation power analysis (CPA) (Brier et al., 2004) are two side channel analysis (or attack) methods. They recover the key by exploiting the statistical dependency between the power consumption of cryptographic devices and the intermediate values inside the ongoing cryptographic algorithm. Intermediate values can be mapped into simulated power consumption with power models. By evaluating the difference (DPA) or correlation (CPA) between simulated and measured power consumptions, a correct hypothesis of intermediate values can be found. The Hamming distance (HD) model and Hamming weight (HW) model are two commonly used power models. Whether the power model can effectively determine the dependency strongly influences the final success rate of power analysis.

‡ Corresponding authors

* Project supported by the Major Program of the Ministry of Industry and Information Technology of China (No. 2017ZX01030301) and the Beijing Natural Science Foundation of China (No. 4162053)

ORCID: Ye YUAN, http://orcid.org/0000-0003-2421-2940

Side channel attacks on HMAC have been discussed in the literature. For example, McEvoy et al. (2007) proposed DPA attack procedures in HD models for HMAC implementation based on the SHA2 family. Belaïd et al. (2015) presented a new DPA strategy in HW models for HMAC-SHA1 and HMAC-SHA2. Even if some hints for attacking HMAC-SM3 could be found from these attack methods, specific analysis methods for HMAC-SM3 are required due to the different structures of the SM3 algorithm.

Guo et al. (2015) presented a DPA strategy for HMAC-SM3 software implementation. Their work is, to our knowledge, the first to deal with HMAC-SM3 side channel security. The 256-bit secret value can be recovered by eight consecutive DPA attacks and several subsequent calculations. Each DPA attack can recover one 32-bit secret intermediate value. Sun et al. (2015) proposed an attack strategy against HMAC-SM3 hardware implementation. Although this work has assumed too many intermediates as register values, almost as many as they wanted, it can also provide some hints for attacking HMAC-SM3 hardware. However, when it comes to hardware implementation, something difficult occurs. It seems quite easy because we just need to recover the same set of intermediate values (eight in total) as those in this literature (Guo et al., 2015). However, in a practical situation, this is complicated, because the power leakage model of HMAC-SM3 hardware implementation is totally different from that of software implementation.

In this study, we deal with one-round-per-cycle hardware implementation (Qu et al., 2015). The eight intermediate values mentioned above are classified into two groups and specific attack strategies are designed for each group. The first group is the easier part to deal with, just like that in Guo et al. (2015). The second group is more complicated. A tricky bit-wise chosen-plaintext attack strategy is proposed for the second group beyond the word-wise chosen-plaintext strategy proposed in Guo et al. (2015). Bit-wise here refers to choosing each bit of one plaintext word as zero or random. By putting the easy and tricky parts together, we achieve one procedure for attacking the whole key of HMAC-SM3 hardware implementation. We also evaluate the proposed method on a field programmable gate array (FPGA) board.

2 SM3 hash algorithm and hardware implementation

The SM3 algorithm (SCA, 2010; Guo et al., 2015) turns an l-bit ($l < 2^{64}$) message into a 256-bit hash value. It includes two main steps: message stuffing and compression iteration. In each iteration, compression is executed once, and a 512-bit message block is expanded into multiple 32-bit plaintext words as inputs for the compression function. To comply with the general concept in side channel analysis, a message is called plaintext here. The total number of compression iterations is determined by the number of blocks after message stuffing, and the i^{th} iteration is described in Algorithm 1.

Algorithm 1 The i^{th} compression iteration

Input: message block B_i, chaining value V_i
Output: next chaining value V_{i+1}

$(W_0, W_1, \dots, W_{15}) \leftarrow B_i$
for $j = 16$ to 67 **do**
  $W_j \leftarrow P_1(W_{j-16} \oplus W_{j-9} \oplus (W_{j-3} \lll 15)) \oplus (W_{j-13} \lll 7) \oplus W_{j-6}$
end for
for $j = 0$ to 63 **do**
  $W_j' \leftarrow W_j \oplus W_{j+4}$
end for
$(A_0, B_0, C_0, D_0, E_0, F_0, G_0, H_0) \leftarrow V_i$
for $j = 0$ to 63 **do**
  $\mathrm{SS1}_j \leftarrow ((A_j \lll 12) + E_j + (T_j \lll j)) \lll 7$
  $\mathrm{SS2}_j \leftarrow \mathrm{SS1}_j \oplus (A_j \lll 12)$
  $\mathrm{TT1}_j \leftarrow \mathrm{FF}_j(A_j, B_j, C_j) + D_j + \mathrm{SS2}_j + W_j' \Longrightarrow \mathrm{TT1}_j \leftarrow \theta_j + W_j'$
  $\mathrm{TT2}_j \leftarrow \mathrm{GG}_j(E_j, F_j, G_j) + H_j + \mathrm{SS1}_j + W_j \Longrightarrow \mathrm{TT2}_j \leftarrow \varphi_j + W_j$
  $D_{j+1} \leftarrow C_j$
  $C_{j+1} \leftarrow B_j \lll 9$
  $B_{j+1} \leftarrow A_j$
  $A_{j+1} \leftarrow \mathrm{TT1}_j$
  $H_{j+1} \leftarrow G_j$
  $G_{j+1} \leftarrow F_j \lll 19$
  $F_{j+1} \leftarrow E_j$
  $E_{j+1} \leftarrow P_0(\mathrm{TT2}_j)$
end for
$V_{i+1} \leftarrow (A_{63}, B_{63}, C_{63}, D_{63}, E_{63}, F_{63}, G_{63}, H_{63}) \oplus V_i$

Notations include left assignment ($\leftarrow$), 32-bit XOR ($\oplus$), left rotation ($\lll$), AND ($\wedge$), OR ($\vee$), NOT ($\sim$), and 32-bit modular addition ($+$), as well as some internal functions and constants, such as two j-related Boolean functions FF_j (Eq. (1)) and GG_j (Eq. (2)), two permutation functions P_0 (Eq. (3)) and P_1 (Eq. (4)), and a j-related constant T_j (Eq. (5)).

$$\mathrm{FF}_j(X,Y,Z) \tag{1}$$
$$= \begin{cases} X \oplus Y \oplus Z, & 0 \le i \le 15, \\ (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z), & 16 \le i \le 63, \end{cases}$$
$$\mathrm{GG}_j(X,Y,Z) \tag{2}$$
$$= \begin{cases} X \oplus Y \oplus Z, & 0 \le i \le 15, \\ (X \wedge Y) \vee (\widetilde{X} \wedge Z), & 16 \le i \le 63, \end{cases}$$
$$P_0(X) = X \oplus (X \lll 9) \oplus (X \lll 17), \tag{3}$$
$$P_1(X) = X \oplus (X \lll 15) \oplus (X \lll 23), \tag{4}$$
$$T_j = \begin{cases} \text{0x79CC4519}, & 0 \le i \le 15, \\ \text{0x7A879D8A}, & 16 \le i \le 63. \end{cases} \tag{5}$$

In Algorithm 1, inputs include a 512-bit message block B_i and a 256-bit chaining value V_i, and the output is the next chaining value V_{i+1}. Lines 1–7 show the message expansion, which generates W_j and $W_j{}'$ for the j^{th} ($0 \le j \le 63$) round in the compression function. Lines 8–22 show the compression function, which includes 64 rounds. For the j^{th} round, eight 32-bit words, A_j, B_j, C_j, D_j, E_j, F_j, G_j, and H_j, and two 32-bit plaintext words, W_j and $W_j{}'$, refer to inputs. The outputs of the j^{th} round are denoted as A_{j+1}, B_{j+1}, C_{j+1}, D_{j+1}, E_{j+1}, F_{j+1}, G_{j+1}, and H_{j+1}. In the so-called "one-round-per-cycle" hardware implementation (Qu et al., 2015), A_j is stored in register A before the j^{th} round, and then A_{j+1} becomes the new value in register A after this round. This case is similar for registers B, C, D, E, F, G, and H. The initial values of the eight registers are A_0, B_0, C_0, D_0, E_0, F_0, G_0, and H_0 (defined in line 8). Finally, line 23 shows how to obtain the output.

Based on Algorithm 1, we design the hardware implementation for SM3-MAC. Fig. 1 shows how our hardware implementation processes the single 512-bit message block. Our implementation employs the structure of a single-cycle round function, which means that each round function corresponds to one clock cycle. The data path of the round function is presented in Fig. 2, on which the power analysis attacks depend. Table 1 shows the comparison between our implementation and the state-of-the-art SM3 circuits. As presented in Table 1, when the device is Cyclone, the resource consumption refers to LEs, and when the device is Virtex-5, the resource consumption refers to Slices containing four look-up tables (LUTs) and four flip flops (FFs).

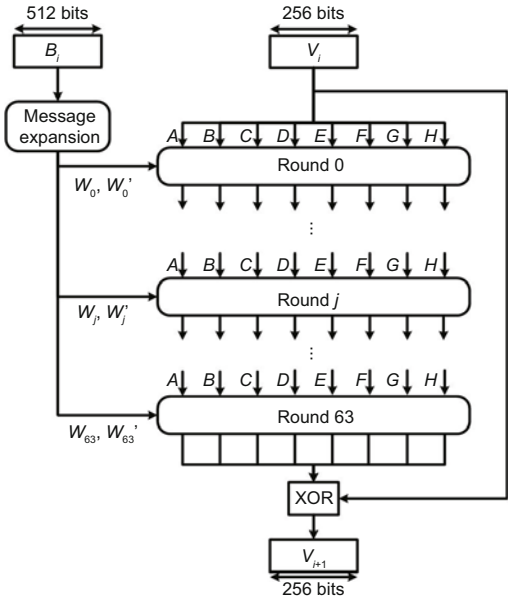

Fig. 1 Process of a single 512-bit message block for SM3-MAC

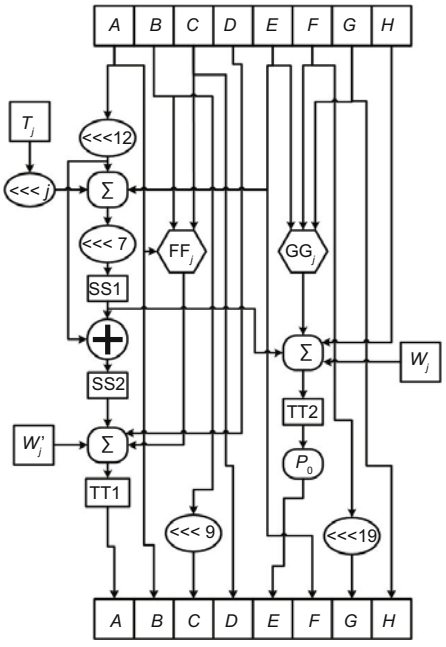

Fig. 2 Data path of the round function

3 Word-wise chosen-plaintext correlation power attack

3.1 Whole picture

In the literature (Kocher et al., 1999; McEvoy et al., 2007; Guo et al., 2015), the attack points for the HMAC algorithm are illustrated, so details will not be reviewed again in this study. The basic concept is that the original HMAC key can never be

recovered by side channel attack methods because it is never combined with a plaintext before a compression function. We recover two constant unknown secret values, which are generated from the original key and some predefined constants. Once the two values are retrieved, the final MAC value can be forged. In this study, we will tackle one of the two secret values, the inner hash value; for simplicity, we call this secret value the key. For the SM3 compression function in Algorithm 1, this key is the input chaining value. It is then segmented into the initial values in eight registers from A through H, and values are A_0, B_0, C_0, D_0, E_0, F_0, G_0, and H_0, respectively, which are what we try to recover. Moradi et al. (2011) successfully performed side-channel attack against bitstream encryption of FPGAs, showing that the 2^{32} combination of power analysis attacks can be possible using graphics processing units (GPUs), which has inspired us with some hints for the proposed attack strategy.

3.1.1 Basic ideas

First, two variables θ_j and φ_j are defined as shown in steps 12 and 13 in Algorithm 1. It can be observed that among all variables in the round function, only θ_j and φ_j are combined with the plaintext in each round. It can also be observed that among all θ_j and φ_j, only θ_0 and φ_0 are constant unknown values which depend on the original keys A_0, B_0, C_0, D_0, E_0, F_0, G_0, and H_0 if the plaintext is purely random, while others are plaintext-controlled variables. So, the plaintext should be chosen carefully to deliberately produce another six constant unknown intermediate values (θ_1–θ_3 and φ_1–φ_3), as shown in Table 2.

In Table 2, T_j $(0 \le j \le 3)$ is simplified to a constant T. The eight unknowns, θ_j $(0 \le j \le 3)$ and φ_j $(0 \le j \le 3)$, are equivalent keys. The simplest case is for θ_0 and φ_0, where all plaintext words are random. In plaintext mode 2, two words, W_0 and W_4, are chosen as zero, while others remain random, and then $W_0{}'$ is apparently zero too. In this way, θ_1 and φ_1 become constant unknown values. Similarly, in plaintext mode 3, two more plaintext words are chosen as zero, and then θ_2 and φ_2 also become constant unknowns. The last case for θ_3 and φ_3 is similar. Because 32-bit plaintext words are chosen as zero or random in a certain manner, we define this way of chosen-plaintext as word-wise.

Actually, the reason that we choose a plaintext like this has already been explained in Guo et al. (2015). A little more will also be explained here. Now suppose that the equivalent keys have already been recovered. Then A_0 can be figured out through the expression of θ_3, and E_0 can be calculated from φ_3. Then from θ_2 and φ_2, B_0 and F_0 can be recovered. Next, it turns to C_0 and G_0. The last ones are D_0 and H_0. In other words, once the equivalent keys are recovered, original keys can be calculated. Actually, the word-wise chosen-plaintext modes listed in Table 2 construct the one-to-one linear mapping

Table 1 Resource consumption comparison

SM3 implementation	Device type	Number of Slices/LEs
This paper	Cyclone	2179 LEs
Wang and Yang (2012)	Cyclone	1789 LEs
Ding and Gao (2012)	Cyclone	1936 LEs
This paper	Virtex-5	433 Slices (1730 LUTs, 1025 FFs)
Liu et al. (2011)	Virtex-5	328 Slices
Ma et al. (2012)	Virtex-5	384 Slices

LE: logic element; LUT: look-up table; FF: flip flop

Table 2 Chosen-plaintext modes and corresponding to-be-attacked intermediate values

No.	Plaintext mode	To-be-attacked intermediate value
1	$W_0, W_1, \ldots, W_{15}$ all random	$\theta_0 = (A_0 \oplus B_0 \oplus C_0) + D_0 + (((A_0 \lll 12) + E_0 + T) \lll 7 \oplus (A_0 \lll 12))$ $\varphi_0 = (E_0 \oplus F_0 \oplus G_0) + H_0 + ((A_0 \lll 12) + E_0 + T) \lll 7$
2	$W_0 = W_4 = 0$, others random	$\theta_1 = (\theta_0 \oplus A_0 \oplus (B_0 \lll 9)) + C_0 + (((\theta_0 \lll 12) + P_0(\varphi_0) + (T \lll 1)) \lll 7 \oplus (\theta_0 \lll 12))$ $\varphi_1 = (P_0(\varphi_0) \oplus E_0 \oplus (F_0 \lll 19)) + G_0 + ((\theta_0 \lll 12) + P_0(\varphi_0) + (T \lll 1)) \lll 7$
3	$W_0 = W_4 = 0$, $W_1 = W_5 = 0$, others random	$\theta_2 = (\theta_1 \oplus \theta_0 \oplus (A_0 \lll 9)) + (B_0 \lll 9) + (((\theta_1 \lll 12) + P_0(\varphi_1) + (T \lll 2)) \lll 7 \oplus (\theta_1 \lll 12))$ $\varphi_2 = (P_0(\varphi_1) \oplus P_0(\varphi_0) \oplus (E_0 \lll 19)) + (F_0 \lll 19) + ((\theta_1 \lll 12) + P_0(\varphi_1) + (T \lll 2)) \lll 7$
4	$W_0 = W_4 = 0$, $W_1 = W_5 = 0$, $W_2 = W_6 = 0$, others random	$\theta_3 = (\theta_2 \oplus \theta_1 \oplus (\theta_0 \lll 9)) + (A_0 \lll 9) + (((\theta_2 \lll 12) + P_0(\varphi_2) + (T \lll 3)) \lll 7 \oplus (\theta_2 \lll 12))$ $\varphi_3 = (P_0(\varphi_2) \oplus P_0(\varphi_1) \oplus (P_0(\varphi_0) \lll 19)) + (E_0 \lll 19) + ((\theta_2 \lll 12) + P_0(\varphi_2) + (T \lll 3)) \lll 7$

between the original keys and equivalent keys.

3.1.2 Complexity in attacking HMAC-SM3 hardware implementation

In Guo et al. (2015), the θ group (including θ_0, θ_1, θ_2, and θ_3) and the φ group (including φ_0, φ_1, φ_2, and φ_3) can be treated in a similar way. DPA attacks toward them are based on Hamming weight leakage of $\mathrm{TT1}_j$ and $\mathrm{TT2}_j$. In one-round-per-cycle hardware implementation (Qu et al., 2015), however, the attack complexity is different. Because power leakage in hardware is mainly due to clocked register transitions, we can construct only Hamming distance models of registers. Note that $\mathrm{TT1}_j$ and $\mathrm{TT2}_j$ are not in registers.

For the θ group, there are four CPA attacks (Table 3). θ_0 is combined with plaintext $W_0{}'$ and generates A_1, which then replaces the former value A_0 and updates register A. This transition will leak power consumption which is correlated with the Hamming distance between A_0 and A_1. So, the power model of CPA 1 is $\mathrm{HD}(A_0, A_1)$. Because both A_0 and θ_0 are unknown and only $W_0{}'$ is controllable, the hypothesis of A_0 and θ_0 should be made simultaneously in the 2^{64} space. When it comes to CPA 3, the key hypothesis space reduces to 2^{32}, because θ_0 has already been recovered after CPA 1. The cases for θ_2 and θ_3 are similar to that for θ_1. So, the implementation order should be CPA 1, CPA 3, CPA 5, and CPA 7.

In parallel, another four CPA attacks for the φ group are shown in Table 3, which are really similar to the four CPA attacks toward the θ group except for the extra P_0 function in their power models. Actually, this linear permutation function makes a lot of difference in practice.

Another thing that should be considered is the computational complexity. All operations in the SM3 compression function are based on 32-bit words. The hypothesis space of 2^{32}, or even 2^{64}, is too huge. In real attack scenarios, to shrink the key space of one single attack, each attack is divided into several partial attacks (Brier et al., 2004; Tunstall et al., 2007), which is the so-called divide-and-conquer method. Typically, a key is divided into parallel sections, for example, by bytes, which is employed in attacking the θ group. However, the divide-and-conquer method for the φ group is much more complicated. The P_0 function contributes to this. To deal with it, a bit-wise chosen-plaintext strategy is proposed for the φ group beyond the word-wise chosen-plaintext strategy.

3.2 Attacking the θ group using the parallel divide-and-conquer method

There are two kinds of attacks for the θ group according to the size of the key hypothesis space. The first kind is CPA 1, and the second kind includes CPA 3, CPA 5, and CPA 7. The key hypothesis spaces of the two kinds are 2^{64} and 2^{32}. In this subsection, we will demonstrate principles and results of both kinds by showing details of CPA 1 and CPA 3.

3.2.1 The first kind: CPA 1

The power model of CPA 1 is $\mathrm{HD}(A_0, A_1)$. In practice, CPA 1 should be decomposed into eight partial-CPA attacks, and its power model should be decomposed into eight corresponding sub-power models as well. The divide-and-conquer method is shown in Fig. 3. In the figure, the numbers from 31 to 0 (below we write as "31:0") represent bits of θ_0 from the most significant bit (MSB) to the least significant bit (LSB). The red bits are unknown, and the black bits are controllable. First, θ_0 modular adds $W_0{}'$, whose result is A_1. Then the sub-power models can be obtained, each of which has Hamming distance between every four bits of A_0 and A_1. As there is

Table 3 Details of each word-wise chosen-plaintext CPA attack

CPA No.	Chosen-plaintext mode	Time range for attack	Intermediate value(s)	Expression for power models
CPA 1	Mode 1	Round 0	θ_0, A_0	$\mathrm{HD}(A_0, A_1) = \mathrm{HW}(A_0 \oplus (\theta_0 + W_0{}'))$
CPA 2	Mode 1	Round 0	φ_0, E_0	$\mathrm{HD}(E_0, E_1) = \mathrm{HW}(E_0 \oplus P_0(\varphi_0 + W_0))$
CPA 3	Mode 2	Round 1	θ_1	$\mathrm{HD}(A_1, A_2) = \mathrm{HW}(\theta_0 \oplus (\theta_1 + W_1{}'))$
CPA 4	Mode 2	Round 1	φ_1	$\mathrm{HD}(E_1, E_2) = \mathrm{HW}(P_0(\varphi_0) \oplus P_0(\varphi_1 + W_1))$
CPA 5	Mode 3	Round 2	θ_2	$\mathrm{HD}(A_2, A_3) = \mathrm{HW}(\theta_1 \oplus (\theta_2 + W_2{}'))$
CPA 6	Mode 3	Round 2	φ_2	$\mathrm{HD}(E_2, E_3) = \mathrm{HW}(P_0(\varphi_1) \oplus P_0(\varphi_2 + W_2))$
CPA 7	Mode 4	Round 3	θ_3	$\mathrm{HD}(A_3, A_4) = \mathrm{HW}(\theta_2 \oplus (\theta_3 + W_3{}'))$
CPA 8	Mode 4	Round 3	φ_3	$\mathrm{HD}(E_3, E_4) = \mathrm{HW}(P_0(\varphi_2) \oplus P_0(\varphi_3 + W_3))$

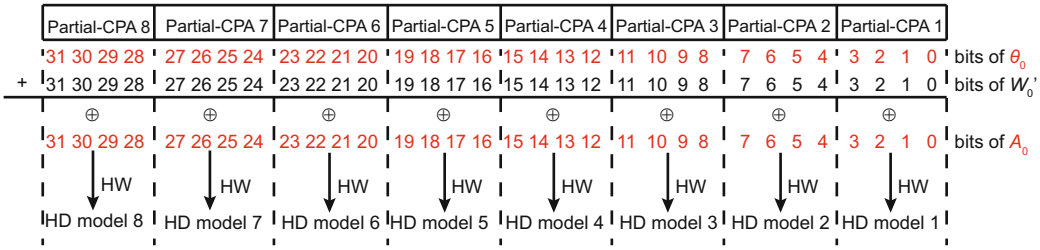

Fig. 3 Divide-and-conquer method of CPA 1 (References to color refer to the online version of this figure)

carry propagation in modular addition, the partial-CPA attacks are not independent. They should be executed from right to left, one by one.

For partial-CPA 1, $\theta_0(3:0)$ and $A_0(3:0)$ are unknown, and $W_0{}'(3:0)$ is the plaintext. Accordingly, the Hamming distance power model is $\mathrm{HD}(A_1(3:0), A_0(3:0)) = \mathrm{HW}\Big(\big(\theta_0(3:0) + W_0{}'(3:0)\big) \oplus A_0(3:0)\Big)$. An attack has been performed on an FPGA board. The results are shown in Fig. 4. The horizontal axis is 2^8 hypothesis of $\theta_0(3:0)||A_0(3:0)$, where "||" means concatenation of two bit vectors. The vertical axis is the largest correlation coefficient between power traces and the computed power model vector based on each key hypothesis. Three features can be observed on the blue waveform.

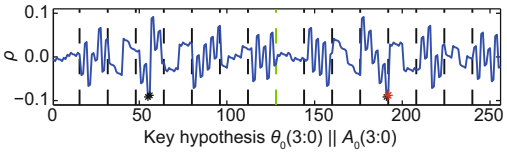

Fig. 4 Results of partial-CPA 1 toward θ_0 (References to color refer to the online version of this figure)

First, it has a pseudo period of 2^7. The green line is the boundary. If we have two key guesses with the same $\theta_0(2:0)$ and $A_0(2:0)$ but opposite $\theta_0(3)$ and opposite $A_0(3)$, for example, "10111111" and "00110111," we will obtain two of the same correlation coefficients. The two key guesses have a difference of $2^7 \pm 2^3$.

Second, one negative correlation coefficient always has one positive counterpart with the same absolute value. In principle, this pair corresponds to two key guesses with the same $\theta_0(3:0)$ and opposite $A_0(3:0)$, for example, "10111111" and "10110000." The waveform is divided into 16 segments. The vertical dotted lines are boundaries between them. Key guesses in each segment have the same $\theta_0(3:0)$, with all possible $A_0(3:0)$. So, one correlation coefficient pair always locates in the same segment. Based on our previous measurement of the register HD leakage from the FPGA board, it should be negative. So, the correct key guess will be looked up only in the negative half of the waveform.

Based on the two features, it can be guessed that there should be two negative peaks corresponding to two candidates for the correct key guess. However, another two very close secondary peaks can be noticed on the waveform, which cannot be neglected. The peaks and secondary peaks are marked by black and red stars. In fact, the red star corresponds to the correct key guess, which will be verified later.

It can be seen that the secondary peaks locate in the same segments as the peaks. As A_0 will be calculated after all eight CPA attacks are finished, the target at this stage is the recovery of $\theta_0(3:0)$. In other words, we need to find which segment (associated with $\theta_0(3:0)$) gives negative peaks instead of differentiating the peaks and secondary peaks located in the same segments. Based on the above analysis, due to the pseudo period feature, there are two candidate values (MSB uncertain) that cannot be distinguished at this moment for the correct guess of $\theta_0(3:0)$. Actually, all the above three features are related to the Hamming distance power model for hardware implementation. In the Hamming weight model, there is no $\oplus A_0(3:0)$ term, so the candidate for the correct key guess is unique.

Based on partial-CPA 1, partial-CPA 2 can be done. Its power model is $\mathrm{HD}(A_1(7:4), A_0(7:4)) = \mathrm{HW}\Big(\big((\theta_0(7:4)||\theta_0(3:0)) + W_0{}'(7:0)\big)(7:4) \oplus A_0(7:4)\Big)$, with $\theta_0(7:4)$ and $A_0(7:4)$ unknown, $W_0{}'(7:0)$ being the plaintext, and $\theta_0(3:0)$ having two candidates. For each candidate, power model vectors can be built and their corresponding correlation coefficient waveform can be acquired. FPGA

attack results of partial-CPA 2 are shown in Fig. 5. The results can be seen in two aspects. First, as the negative peaks appear in candidate 2 rather than the candidate 1 sub-figure, it can be affirmed that the correct guess of $\theta_0(3:0)$ is candidate 2. The reason is that the carry from bit 3 to bit 4 in the modular addition in the power model of partial-CPA 2 can be correct only when $\theta_0(3:0)$ is correct. Second, there are two noticeable negative peaks which correspond to two candidate $\theta_0(7:4)$ guesses with their MSBs uncertain.

Similar to the role that $\theta_0(3:0)$ plays in partial-CPA 2, $\theta_0(7:4)$ will play a role in partial-CPA 3 whose power model is $\mathrm{HD}(A_1(11:8), A_0(11:8)) = \mathrm{HW}\Big(((\theta_0(11:8)||\theta_0(7:4)||\theta_0(3:0)) + W_0'(11:0))(11:8) \oplus A_0(11:8)\Big)$, where $\theta_0(3:0)$ is known after partial-CPA 2 and $\theta_0(7:4)$ has two candidates. The results of partial-CPA 3 are shown in Fig. 6. In a similar way, partial-CPA 4 to partial-CPA 8 can be done one by one. Each partial-CPA uses the same set of power traces in word-wise chosen-plaintext mode 1 listed in Table 3, and depends on the results of all former partial-CPA attacks. As partial-CPA 8 is the last one and no one can distinguish its two key candidates (MSB uncertain), there are two candidates for θ_0 (MSB uncertain).

3.2.2 The second kind: CPA 3 as an example

According to Table 3, the word-wise chosen-plaintext mode of CPA 3 is mode 2, and θ_1 is the to-be-attacked value. To maintain each partial-CPA with the same key hypothesis space of 2^8, here CPA 3 is decomposed into four partial-CPA attacks (Fig. 7).

For partial-CPA 1 of CPA 3, the power model is $\mathrm{HD}(A_2(7:0), A_1(7:0)) = \mathrm{HW}\Big((\theta_1(7:0) + W_1'(7:0)) \oplus \theta_0(7:0)\Big)$, where $\theta_1(7:0)$ is unknown and $W_1'(7:0)$ is random. The FPGA attack results are shown in Fig. 8. The waveform does not have symmetrical and pseudo period features like Fig. 4, as the XORed term $\theta_0(7:0)$ is a constant value. So, in theory the correct guess for $\theta_1(7:0)$ is unique. In practice, there is luckily a single peak in Fig. 8. In terms of partial-CPA 2, its power model is based on the hypothesis of $\theta_1(15:8)$, whose experimental results are shown in Fig. 9. Due to the interference of noise, there is a secondary peak not far from the peak, which cannot be abandoned, leading the two candidates of $\theta_1(15:8)$ into partial-CPA 3.

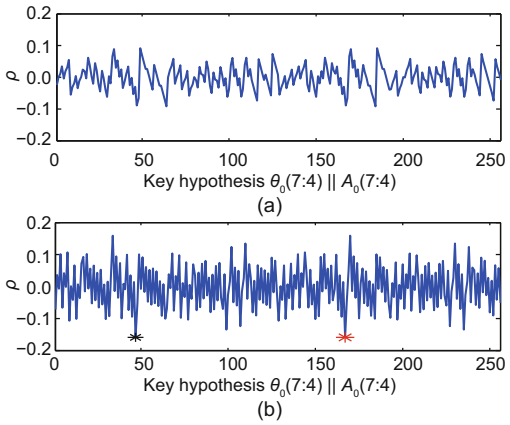

Fig. 5 Results of partial-CPA 2 toward θ_0: (a) candidate 1; (b) candidate 2

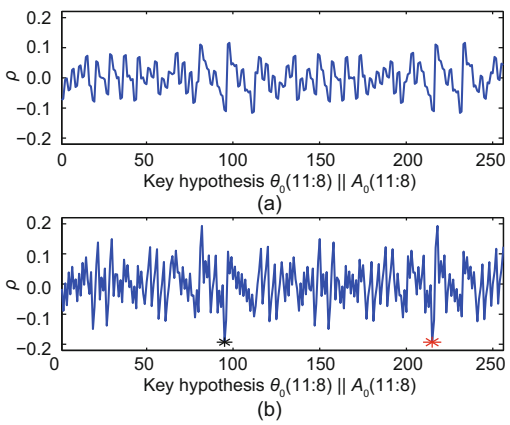

Fig. 6 Results of partial-CPA 3 toward θ_0: (a) candidate 1; (b) candidate 2

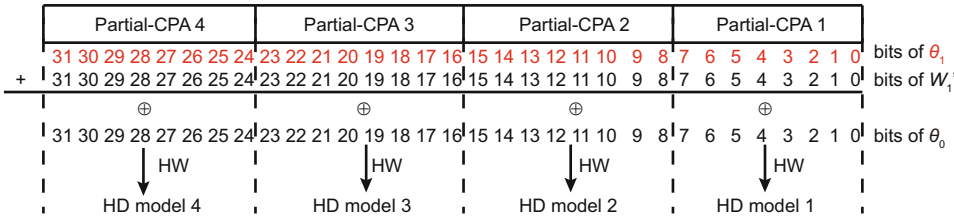

Fig. 7 Divide-and-conquer method of CPA 3

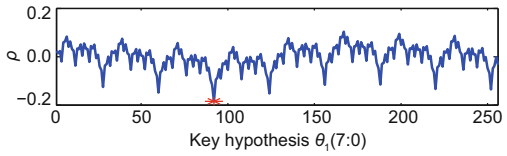

Fig. 8 Results of partial-CPA 1 toward θ_1

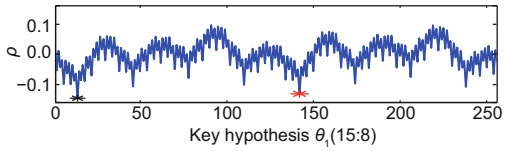

Fig. 9 Results of partial-CPA 2 toward θ_1

As shown in Fig. 10, candidate 2 is the true $\theta_1(15:8)$, and the single peak in the candidate 2 sub-figure reveals the correct guess for $\theta_1(23:16)$. Whenever more peaks appear than expected in one partial-CPA attack, they just need to be plugged into the subsequent partial-CPA attack to remove the confusing incorrect ones, which is a good way to deal with noise, especially when the power model is linear, as used in this study. Finally, partial-CPA 4 is performed twice under the assumption of $\theta_0(31)=0$ and $\theta_0(31)=1$, whose results are shown in Fig. 11. According to the figure, there is one peak marked in red in each sub-figure. The two peaks correspond to two key guesses with the same $\theta_1(30:24)$ and opposite $\theta_1(31)$, and only one of them is true, but the correct one cannot be distinguished now. So, the conclusion is that there are also two candidates for θ_1 with uncertain MSB, but they have a fixed relationship with the two candidates for θ_0.

CPA 5 and CPA 7 are exactly in the same procedure as CPA 3. Just replace the variables and use their own corresponding word-wise chosen-plaintext modes. Both θ_2 and θ_3 will have two candidates with MSB uncertain, but their relationship with the two candidates of θ_0 and θ_1 is fixed. Therefore, there are two candidate groups for the θ group.

4 Bit-wise chosen-plaintext attack: attacking the φ group

As shown in Table 3, CPA 2, CPA 4, CPA 6, and CPA 8 are four subsequent CPA attacks toward the φ group. In the first subsection, the first kind of attack, CPA 2, will be introduced. The second subsection introduces the second kind by showing the details of CPA 4. As mentioned, due to the extra

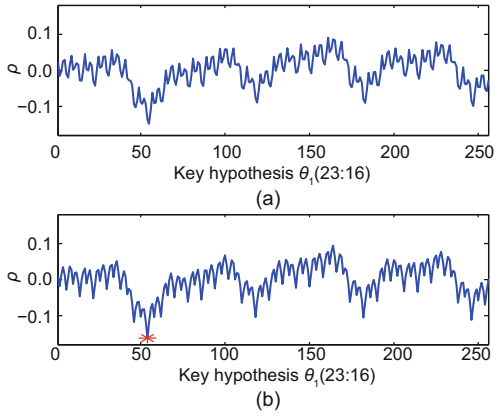

Fig. 10 Results of partial-CPA 3 toward θ_1: (a) candidate 1; (b) candidate 2

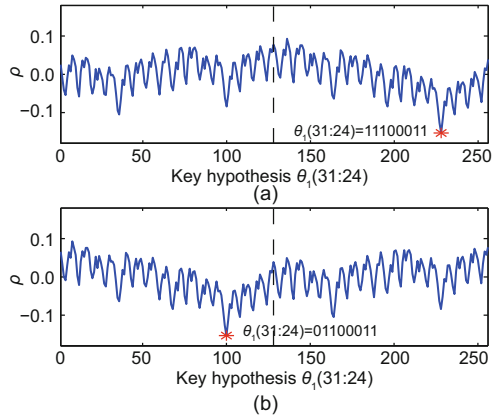

Fig. 11 Results of partial-CPA 4 toward θ_1 assuming $\theta_0(31)=0$ (a) and $\theta_0(31)=1$ (b)

P_0 permutation function, the φ group is much more complicated to deal with compared with the θ group.

4.1 The first kind: CPA 2

As listed in Table 3, the power model of CPA 2 is $\mathrm{HD}(E_0,E_1)=\mathrm{HW}(E_0\oplus P_0(\varphi_0+W_0))=\mathrm{HW}(E_0\oplus P_0(\mathrm{TT2}_0))$, where $\mathrm{TT2}_0=\varphi_0+W_0$. The operations in this power model are shown in Fig. 12, in which the numbers $(31:0)$ above the horizontal line represent bits of $\mathrm{TT2}_0$. $\mathrm{TT2}_0$, $\mathrm{TT2}_0\lll 9$, and $\mathrm{TT2}_0\lll 17$ are XORed together according to the P_0 function in Eq. (3) and results in E_1. Then the Hamming distance between E_1 and E_0 can be calculated. As mentioned, CPA 2 should also be decomposed into several partial-CPA attacks, but the divide-and-conquer method is totally a new thing. Due to permutation among bits of $\mathrm{TT2}_0$, the key space cannot be simply divided into parallel sections like in Section 3. How CPA 2 is decomposed is shown in Fig. 12, whose

details will be demonstrated later.

Fig. 13 shows the structure of the φ_0 modular adding W_0. Obviously, there is carry propagation from lower to higher bits. It can be observed from Fig. 12 that to use HD leakage on the lower two bits of register E and build a power model as $\mathrm{HD}(E_0(1:0), E_1(1:0)) = \mathrm{HW}\Big((\mathrm{TT2}_0(1:0) \oplus \mathrm{TT2}_0(16:15) \oplus \mathrm{TT2}_0(24:23)) \oplus E_0(1:0)\Big)$, six bits of $\mathrm{TT2}_0$ including $\mathrm{TT2}_0(1:0)$, $\mathrm{TT2}_0(16:15)$, and $\mathrm{TT2}_0(24:23)$ should be determined. As presented in Fig. 13, $\mathrm{TT2}_0(1:0)$ is easy to obtain, and depends on the hypothesis of $\varphi_0(1:0)$ and the controllable $W_0(1:0)$. Because there are so many unknown bits of φ_0, how to determine all six bits of $\mathrm{TT2}_0$ inside the three boxes becomes the major problem.

Now we discuss our bit-wise chosen-plaintext mode for partial-CPA 1. As shown in Fig. 13, except the random bits marked as r, all other bits of W_0 are selected as "0." It is noted that W_0 is just random in the word-wise chosen-plaintext mode for CPA 2. The to-be-recovered bits of φ_0 in partial-CPA 1 are marked as "?" and there are 2^6 hypotheses of them. Other bits of φ_0 are marked as "*." They are also unknown bits, but they are not the to-be-recovered bits in partial-CPA 1. Unless $\varphi_0(14:2)$ is a vector of "1," there must be no carry from bit 14 to bit 15. When $\varphi_0(14:2)$ is a vector of "1," whether there is carry from bit 14 to bit 15 depends on $\varphi_0(1:0)$ and $W_0(1:0)$. In other words, the carry from bit 14 to bit 15 is predictable. Then $\mathrm{TT2}_0(16:15)$ can be determined and the carry from bit 16 to bit 17 is also predictable. Similarly, the carry from bit 22 to bit 23 can be predicted and $\mathrm{TT2}_0(24:23)$ can be figured out according to whether $\varphi_0(22:17)$ is a vector of "1" or not. So, there exist four cases for partial-CPA 1 as shown in Table 4. The power model in each case is calculated differently (Table 5).

The key bits of partial-CPA 1, concatenated as an 8-bit vector of $\varphi_0(1:0)||\varphi_0(16:15)||\varphi_0(24:23)||E_0(1:0)$, whose hypothesis space is 2^8, are the

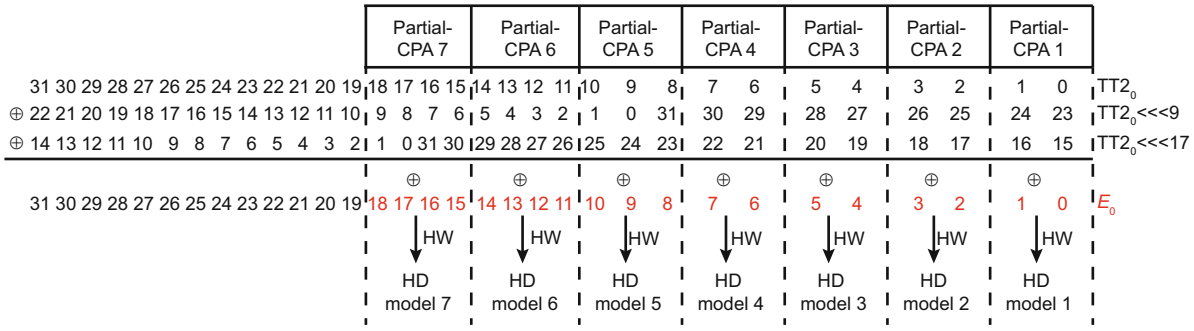

Fig. 12 Divide-and-conquer method of CPA 2

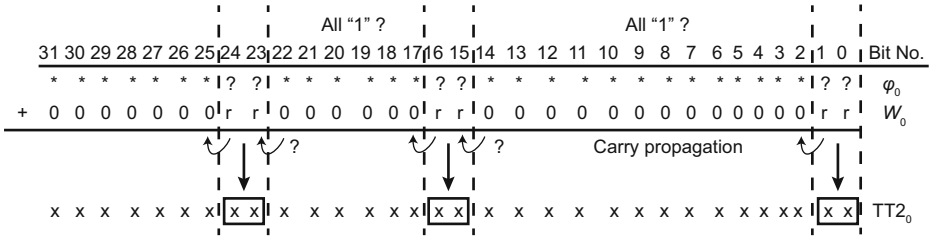

Fig. 13 Bit-wise chosen-plaintext mode for partial-CPA 1 toward φ_0

Table 4 Four cases of partial-CPA 1 in CPA 2

Case	$\varphi_0(14:2)$ vector of "1"?	$\varphi_0(22:17)$ vector of "1"?	Carry from bit 14 to bit 15?	Carry from bit 22 to bit 23?
A	No	No	Never	Never
B	No	Yes	Never	Possible
C	Yes	No	Possible	Never
D	Yes	Yes	Possible	Possible

Table 5 Power models in four cases of partial-CPA 1 in CPA 2

Case	Power model of partial-CPA 1
A	$\mathrm{HW}\Big(\big(\varphi_0(1:0)+W_0(1:0)\big)\oplus\big(\varphi_0(16:15)+W_0(16:15)\big)\oplus\big(\varphi_0(24:23)+W_0(24:23)\big)\oplus E_0(1:0)\Big)$
B	$\mathrm{HW}\Big(\big(\varphi_0(1:0)+W_0(1:0)\big)\oplus\big(\varphi_0(24:15)+W_0(24:15)\big)(1:0)\oplus\big(\varphi_0(24:15)+W_0(24:15)\big)(9:8)\oplus E_0(1:0)\Big)$
C	$\mathrm{HW}\Big(\big(\varphi_0(16:0)+W_0(16:0)\big)(1:0)\oplus\big(\varphi_0(16:0)+W_0(16:0)\big)(16:15)\oplus\big(\varphi_0(24:23)+W_0(24:23)\big)\oplus E_0(1:0)\Big)$
D	$\mathrm{HW}\Big(\big(\varphi_0(24:0)+W_0(24:0)\big)(1:0)\oplus\big(\varphi_0(24:0)+W_0(24:0)\big)(16:15)\oplus\big(\varphi_0(24:0)+W_0(24:0)\big)(24:23)\oplus E_0(1:0)\Big)$

same as the partial-CPA attacks towards the θ group. This is the reason why partial-CPA 1 is done based on two bits of register E.

Partial-CPA 1 attack experiments were successfully performed in four cases on an FPGA board. As shown in Fig. 14, eight negative peaks appear in case A, and the four peaks in case B are their subset. It is almost impossible to confirm which case is correct as the magnitudes of peaks in cases A and B have little difference. Actually, there is no need to recognize the correct case; the only thing to do is to find all candidates for the correct key hypothesis corresponding to the peaks in all the four cases. There are eight candidates for $\varphi_0(1:0)||\varphi_0(16:15)||\varphi_0(24:23)$. The true one will be determined in partial-CPA 2.

As shown in Fig. 12, partial-CPA 2 uses HD leakage on bit 3 and bit 2 of register E, so $\mathrm{TT2}_0(3:2)$, $\mathrm{TT2}_0(18:17)$, and $\mathrm{TT2}_0(26:25)$ should be predictable. To obtain the six bits, a specific bit-wise chosen-plaintext mode for partial-CPA 2 is needed (Fig. 15), where the bits referred to $\varphi_0(1:0)||\varphi_0(16:15)||\varphi_0(24:23)$ are recovered with eight candidates. Every two "?" bits before the "$\surd$" bits are to-be-recovered bits of φ_0 in partial-CPA 2. Considering another two key bits $E_0(3:2)$, the key hypothesis space is also 2^8. In plaintext W_0, there are six more random bits compared to W_0 in partial-CPA 1. For partial-CPA 2, there are also four cases. The definition of each case is similar to that in Table 4, with $\varphi_0(14:2)$ replaced by $\varphi_0(14:4)$ and $\varphi_0(22:17)$ replaced by $\varphi_0(22:19)$. In each case, the carries from bit 14 to bit 15 and from bit 22 to bit 23 can be predicted, and the six new bits of $\mathrm{TT2}_0$ in the new boxes can be determined to calculate the power model as listed in Table 6.

A partial-CPA 2 attack on an FPGA board has been performed successfully, and the results are shown in Fig. 16. It can be seen that the difference between the cases in the same column is relatively small, while the difference between the candidates in the same row is rather large. Luckily, our goal is to find the true candidate rather than caring about the cases, which does not mean that there is no point in building the power models in different cases at the beginning. Obviously, candidate 8 for $\varphi_0(1:0)||\varphi_0(16:15)||\varphi_0(24:23)$ is the correct one because it results in the largest peaks in all the four cases compared with other candidates. In addition, because there are again eight peaks in each sub-figure of candidate 8, there are eight candidates for $\varphi_0(3:2)||\varphi_0(18:17)||\varphi_0(26:25)$.

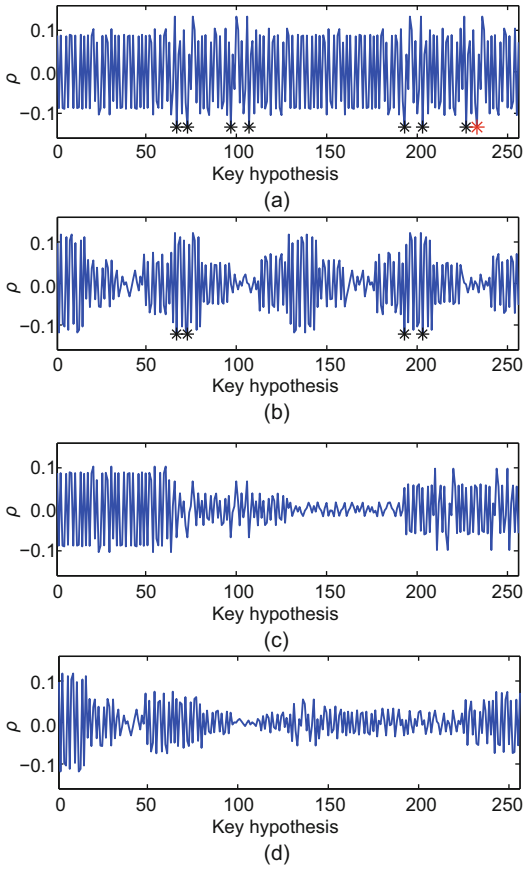

Fig. 14 Results of partial-CPA 1 toward φ_0: (a) case A; (b) case B; (c) case C; (d) case D

The partial-CPA attacks are introduced as follows. For each new partial-CPA attack, new

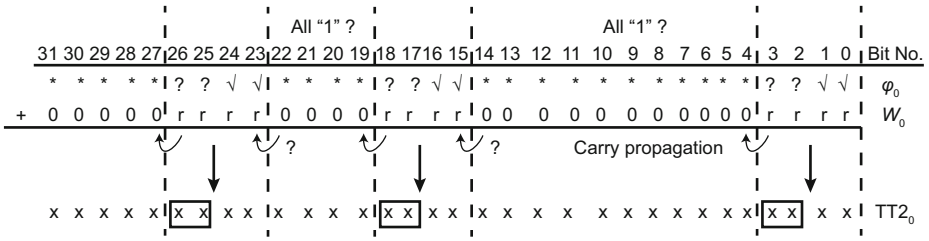

Fig. 15 Bit-wise chosen-plaintext mode for partial-CPA 2 toward φ_0

Table 6 Power models in four cases of partial-CPA 2 in CPA 2

Case	Power model of partial-CPA 2
A	$\mathrm{HW}\Big(\big(\varphi_0(3:0)+W_0(3:0)\big)(3:2)\oplus\big(\varphi_0(18:15)+W_0(18:15)\big)(3:2)\oplus\big(\varphi_0(26:23)+W_0(26:23)\big)(3:2)\oplus E_0(3:2)\Big)$
B	$\mathrm{HW}\Big(\big(\varphi_0(3:0)+W_0(3:0)\big)(3:2)\oplus\big(\varphi_0(26:15)+W_0(26:15)\big)(3:2)\oplus\big(\varphi_0(26:15)+W_0(26:15)\big)(11:10)\oplus E_0(3:2)\Big)$
C	$\mathrm{HW}\Big(\big(\varphi_0(18:0)+W_0(18:0)\big)(3:2)\oplus\big(\varphi_0(18:0)+W_0(18:0)\big)(18:17)\oplus\big(\varphi_0(26:23)+W_0(26:23)\big)(3:2)\oplus E_0(3:0)\Big)$
D	$\mathrm{HW}\Big(\big(\varphi_0(26:0)+W_0(26:0)\big)(3:2)\oplus\big(\varphi_0(26:0)+W_0(26:0)\big)(18:17)\oplus\big(\varphi_0(26:0)+W_0(26:0)\big)(26:25)\oplus E_0(3:0)\Big)$

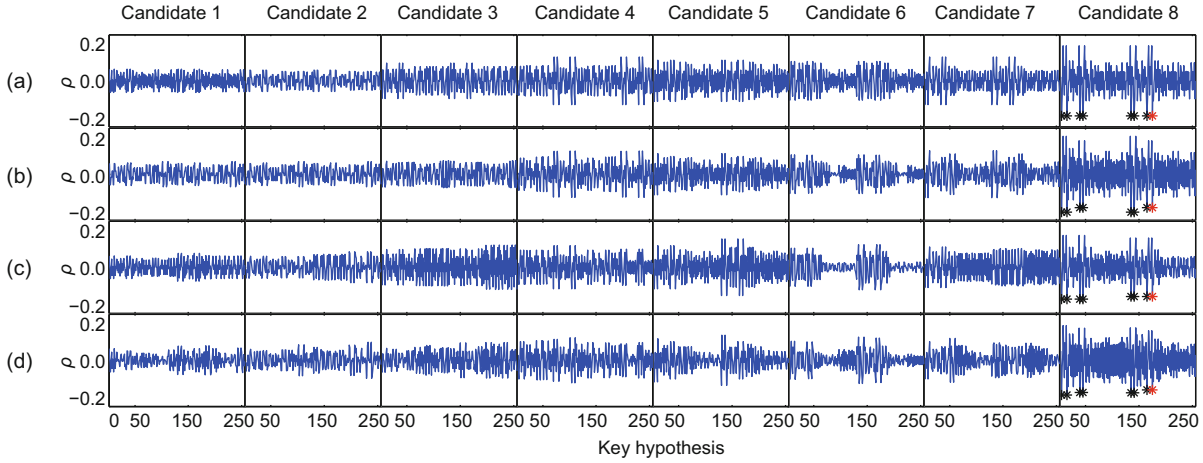

Fig. 16 Results of partial-CPA 2 toward φ_0: (a) case A; (b) case B; (c) case C; (d) case D

Each row represents one case of the power model of partial-CPA 2, while each column corresponds to one candidate for $\varphi_0(1:0)||\varphi_0(16:15)||\varphi_0(24:23)$ from the results of partial-CPA 1. In each sub-figure, the horizontal axis represents the value of the key hypothesis denoted by $\varphi_0(3:2)||\varphi_0(18:17)||\varphi_0(26:25)||E_0(3:2)$, and the vertical axis is the correlation coefficient ρ

to-be-recovered bits on the left of the recently recovered bits by the former partial-CPA attack are defined. At the same time, more bits on the adjacent left in the plaintext word W_0 are added as random bits. For example, as shown in Fig. 17, for partial-CPA 3, $\varphi_0(3:2)$, $\varphi_0(18:17)$, and $\varphi_0(26:25)$ are recovered (eight candidates), and the new to-be-recovered bits are just neighbor bits on their left. There are also four cases of partial-CPA 3. Partial-CPA 4 is similar, but there are only two cases (Fig. 18), because only one unknown middle part of φ_0 is left and should be assumed as a vector of "1" or not. The results of partial-CPA 3 and partial-CPA 4 are shown in Figs. 19 and 20, respectively. These figures show that partial-CPA 3 confirms candidate 8 from partial-CPA 2 as the correct one, and partial-CPA 4 confirms candidate 6 from partial-CPA 3 as the correct one.

As shown in Fig. 12, partial-CPA 1–4 all use HD leakage on every two bits of register E. However, from partial-CPA 5–7, things seem to have changed. Actually, as more bits of φ_0 are recovered, HD leakage on more than two bits of register E are used to keep the key space at around 2^8. For a better understanding, we take partial-CPA 5 as an example. In partial-CPA 5, if the HD leakage on only two bits

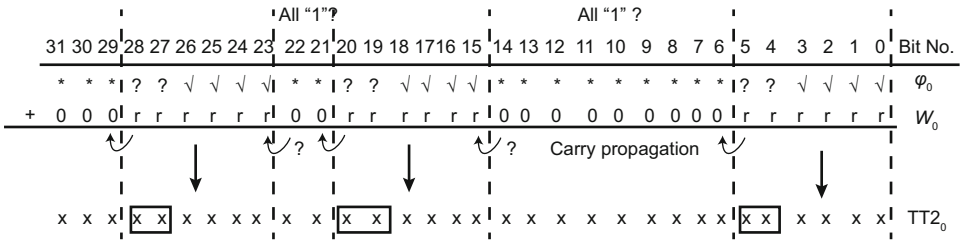

Fig. 17 Bit-wise chosen-plaintext mode for partial-CPA 2 toward φ_0

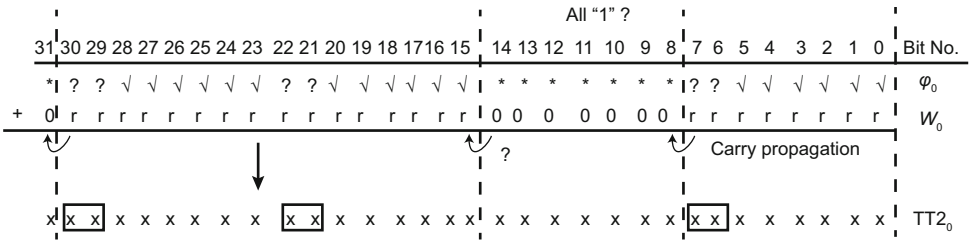

Fig. 18 Bit-wise chosen-plaintext mode for partial-CPA 4 toward φ_0

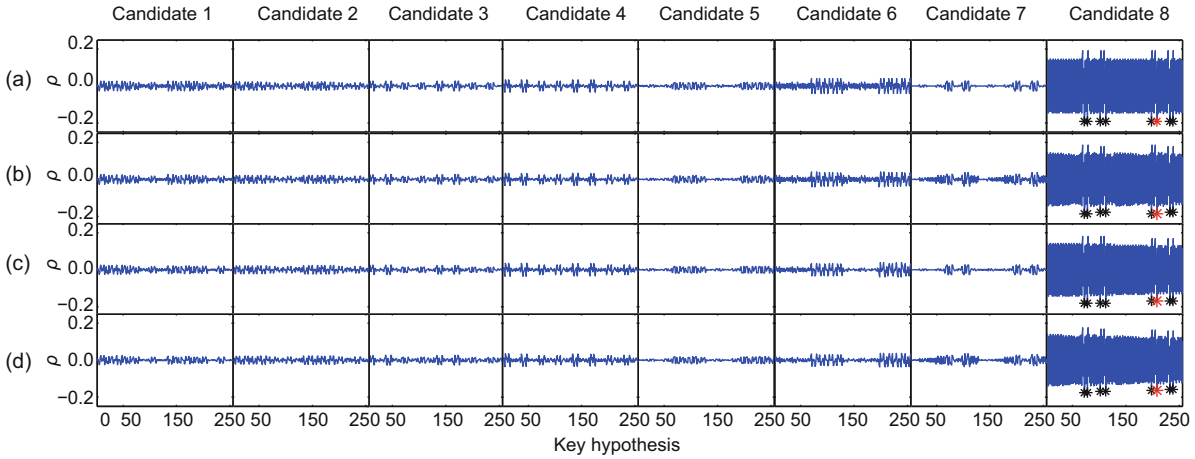

Fig. 19 Results of partial-CPA 3 toward φ_0: (a) case A; (b) case B; (c) case C; (d) case D

Each row represents one case of the power model of partial-CPA 3, while each column corresponds to one candidate for $\varphi_0(3:2)||\varphi_0(18:17)||\varphi_0(26:25)$ from the results of partial-CPA 2. In each sub-figure, the horizontal axis represents the value of the key hypothesis denoted by $\varphi_0(5:4)||\varphi_0(20:19)||\varphi_0(28:27)||E_0(5:4)$, and the vertical axis is the correlation coefficient ρ

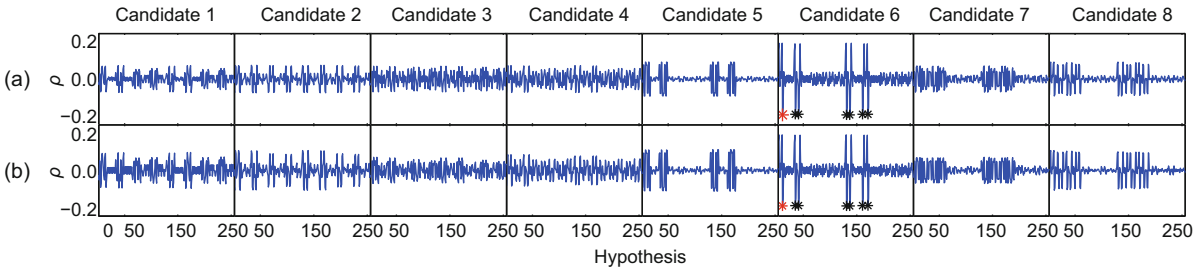

Fig. 20 Results of partial-CPA 4 toward φ_0: (a) case A; (b) case B

(bit 9 and bit 8) of register E is used, the key space will be only 2^5 because $\varphi_0(0)$ and $\varphi_0(24:23)$ are known based on former partial-CPA attacks. However, if leakage on one more bit is used as shown in Fig. 12, the key space will be 2^7; if we use leakage on two more bits, then the key space is 2^9. In this study, the 2^7 choice is selected for partial-CPA 5, whose key is then $\varphi_0(31)||\varphi_0(10:8)||E_0(10:8)$. The bit-wise chosen-plaintext mode for partial-CPA 5 is shown in Fig. 21. Under the assumption that $\varphi_0(14:11)$ is all "1" vector or not, there are also two cases. The experimental results are shown in Fig. 22, indicating that candidate 2 from partial-CPA 4 is the true one. Although there are four negative peaks in the sub-figure of candidate 2, only the two candidates for $\varphi_0(10:8)$ will be distinguished by the following partial-CPA 6. The MSB $\varphi_0(31)$ cannot be distinguished, because the carry generated on bit 31 is discarded and does not influence other bits. Only the bits that can propagate carries to the higher bits can be recovered.

In partial-CPA 6, W_0 has no chosen feature (Fig. 23), and all four remaining unknown bits of φ_0 are to-be-recovered bits. Considering $E_0(14:11)$, the key space is again 2^8. This is why we use leakage on four bits of register E in partial-CPA 6. Experimental results are shown in Fig. 24. Because there are two peaks in the sub-figure of candidate 1, the true value of $\varphi_0(10:8)$ is "011" rather than "111." The values of $\varphi_0(14:11)$ corresponding to the two peaks have only one different bit $\varphi_0(14)$. Because this bit may propagate carry to higher bits, another partial-CPA attack occurs, called partial-CPA 7, with the same set of plaintext as in partial-CPA 6, to recover the single bit. Leakage on bits (15:11) of register E can be used in partial-CPA 6, and we can save partial-CPA 7. However, in this way, the key space of partial-CPA 6 will be 2^9. Attackers can choose either way. Now we have recovered all bits of φ_0 except its MSB $\varphi_0(31)$. In conclusion, the key bits and bit-wise chosen-plaintext modes for each partial-CPA are listed in Table 7.

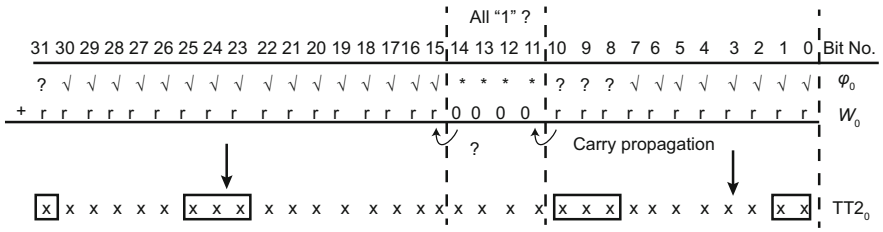

Fig. 21 Bit-wise chosen-plaintext mode for partial-CPA 5 toward φ_0

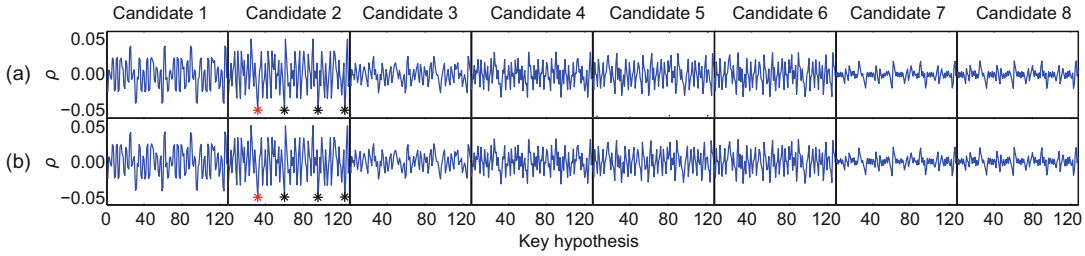

Fig. 22 Results of partial-CPA 5 toward φ_0: (a) case A; (b) case B

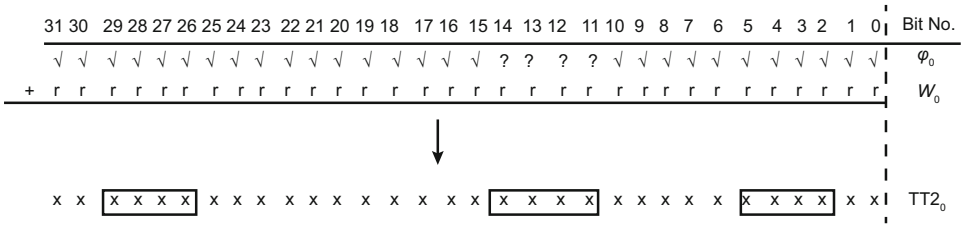

Fig. 23 Bit-wise chosen-plaintext mode for partial-CPA 6 toward φ_0

4.2 The second kind: CPA 4 as an example

In practice, CPA 4 should also be decomposed into several partial-CPA attacks like we did for CPA 2. The divide-and-conquer method is shown in Fig. 25. Note $\mathrm{TT2}_1 = \varphi_1 + W_1$. The keys and bit-wise chosen-plaintext modes of W_1 for each partial-CPA attack are given in Table 8. The basic principles are quite similar to those in CPA 2. Remember that $\varphi_0(31)$ is uncertain after CPA 2. Because bit 8 of $P_0(\varphi_0)$ which is marked red in Fig. 25 is related to $\varphi_0(31)$, it is also uncertain. So, $\varphi_1(31)$ cannot be determined, but one possible $\varphi_1(31)$ corresponds to only one possible $\varphi_0(31)$.

For CPA 6 and CPA 8, the story is quite similar. Thus, two candidate groups for the φ group can be acquired. Combining the two candidate groups for the θ group, four possible original keys will be determined. In other words, the key space can be reduced from 2^{256} to 4 at this stage.

5 Recovery of the unique key

All attack experiments on FPGA boards have been performed successfully. In total, 39 partial-CPA attack experiments have been implemented,

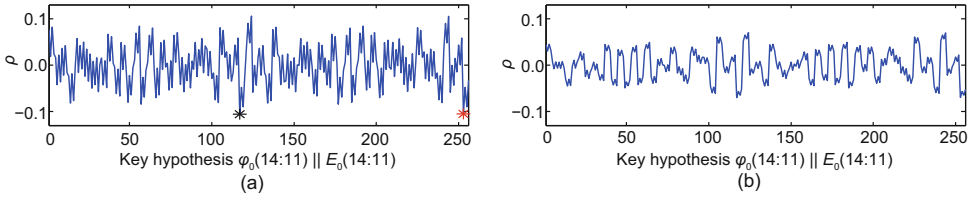

Fig. 24 Results of partial-CPA 6 toward φ_0: (a) candidate 1 ($\varphi_0(10:8) = 011$); (b) candidate 2 ($\varphi_0(10:8) = 111$)

Table 7 Conclusions for important details of all partial-CPA attacks in CPA 2

Partial-CPA No.	Key bits to be recovered	Key space	Bit-wise chosen-plaintext mode of W_0
1	$\varphi_0(1:0)\|\|\varphi_0(16:15)\|\|\varphi_0(24:23)\|\|E_0(1:0)$	2^8	Bits on (24:23), (16:15), and (1:0) random, other bits fixed at "0"
2	$\varphi_0(3:2)\|\|\varphi_0(18:17)\|\|\varphi_0(26:25)\|\|E_0(3:2)$	2^8	Bits on (26:23), (18:15), and (3:0) random, other bits fixed at "0"
3	$\varphi_0(5:4)\|\|\varphi_0(20:19)\|\|\varphi_0(28:27)\|\|E_0(5:4)$	2^8	Bits on (28:23), (20:15), and (5:0) random, other bits fixed at "0"
4	$\varphi_0(7:6)\|\|\varphi_0(22:21)\|\|\varphi_0(30:29)\|\|E_0(7:6)$	2^8	Bits on (30:15) and (7:0) random, other bits fixed at "0"
5	$\varphi_0(31)\|\|\varphi_0(10:8)\|\|E_0(10:8)$	2^7	Bits on (31:15) and (10:0) random, other bits fixed at "0"
6	$\varphi_0(14:11)\|\|E_0(14:11)$	2^8	All bits random

Table 8 Conclusions for important details of all partial-CPA attacks in CPA 4

Partial-CPA No.	Key bits to be recovered	Key space	Bit-wise chosen-plaintext mode of W_0
1	$\varphi_0(2:0)\|\|\varphi_0(17:15)\|\|\varphi_0(25:23)$	2^9	Bits on (25:23), (17:15), and (2:0) random, other bits fixed at "0"
2	$\varphi_0(5:3)\|\|\varphi_0(20:18)\|\|\varphi_0(28:26)$	2^9	Bits on (28:23), (20:15), and (5:0) random, other bits fixed at "0"
3	$\varphi_0(8:6)\|\|\varphi_0(22:21)\|\|\varphi_0(31:29)$	2^8	Bits on (31:15) and (8:0) random, other bits fixed at "0"
4	$\varphi_0(14:9)$	2^6	All bits random

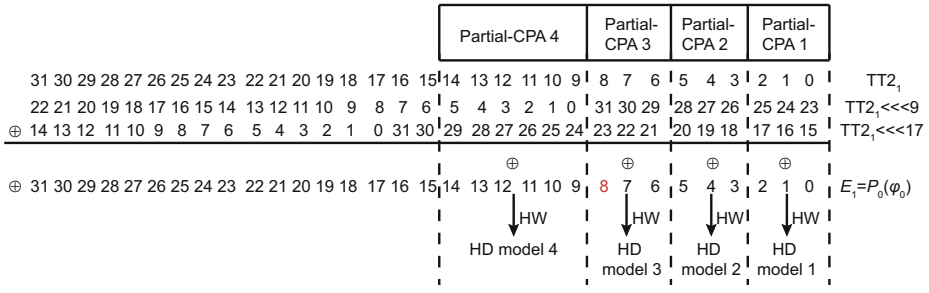

Fig. 25 Divide-and-conquer method of CPA 4 (References to color refer to the online version of this figure)

with each recovering several bits of the equivalent keys. Over 400 000 power traces have been collected and analyzed. Due to the inherent linearity of the attacked operations, power leakage does not differ so much between some key guesses, which can be seen in the above figures showing attack results, so each partial-CPA attack consumes more than 10 000 power traces on average. After all the experiments and some calculations, four possible original keys, which are listed in Table 9, were recovered. Recall that A_0 plays a role in CPA 1 and that E_0 plays a role in CPA 2. Some of their bits can be recovered by every partial-CPA attack, although there is some uncertainty. The recovery results of θ_0 and A_0 in our experiment are given in Table 10. By comparing Tables 9 and 10, we can conclude that the first candidate for the original key is the correct one. Of course the correct candidate can also be found by comparing recovered E_0 and calculated E_0. In other words, any uncertainty due to the inherent linearity of power leakage can be eliminated in this stage.

6 Conclusions

In this paper, an attack procedure has been proposed for the one-round-per-cycle HMAC-SM3 hardware implementation. The prerequisite here is that we assume that only the round registers leak and they leak only their HD but not HW, which is consistent with our real measurement on an FPGA (SAKURA-G) implementation of HMAC-SM3. Because no power leakage models except the round register Hamming distance models were available, the CPA attack on HMAC-SM3 hardware implementation was much more complex than its software counterpart. Based on this prerequisite, the improved word-wise chosen-plaintext attack procedure theory was first given in this paper. By following this procedure, the secret values should be determined by executing eight CPA attacks with a key hypothesis space of 2^{64} or 2^{32}, which is, however, too large and impossible for modern computers to process. In practice, the divide-and-conquer method can be used to decompose each CPA attack into several partial-CPA attacks, each with a much smaller key hypothesis space. According to the different inherent properties of the two groups of secret values in the SM3 algorithm, the methods of divide-and-conquer for them are totally different. One is straightforward, and involves segmenting the long key into parallel shorter subkeys. The other is more trickier, requiring the bit-wise chosen-plaintext CPA attack procedure, which is the main focus of this paper. For the recovery of the whole key, 39 partial-CPA attacks for the two groups were needed, and they should follow the strict sequence in the same group. All partial-CPA attack experiments on FPGA boards have been performed successfully, which verifies the proposed attack scheme. By comparing experimental results with some computation results, the unique key was obtained.

Acknowledgements

The authors would like to thank Xu-guang GUAN, Tao SUN, Yong GU, et al., who are from State Key Laboratory of Cryptography, for the nice cooperation and meaningful suggestion.

Compliance with ethics guidelines

Ye YUAN, Kai-ge QU, Li-ji WU, Jia-wei MA, and Xiang-min ZHANG declare that they have no conflict of interest.

Table 9 Four candidates for the correct key in one attack experiment

No.	Candidate for the correct key
1	7380166F4914B2B9172442D7DA8A0600A96F30BC163138AAE38DEE4DB0FB0E4E
2	F3C0186AC914BAC1974FADDACA5C27CC296F30BD163138A963A1EE4CA0F30B91
3	93FFD62F494FB0B8C0FC00D78F610B8439783FCD76323968620963F45E030312
4	13BFD82BC94FB8C040C8EFDBA0392CD8B9783FCD76323969E20D63F44E230553

Table 10 Recovery results of θ_0 and A_0 in one attack experiment

Bits	θ_0	A_0	Bits	θ_0	A_0	Bits	θ_0	A_0
31:28	0101, 1101	0111(7), 1111(F)	19:16	1011	0000(0), 0001(1)	7:4	1010	0110(6)
27:24	1000	0011(3), 0111(7)	15:12	0101	0001(1)	3:0	1011	1111(F), 1110(E)
23:20	1000	1000(8)	11:8	1101	0110(6)			

References

Belaïd S, Bettale L, Dottax E, et al., 2015. Differential power analysis of HMAC SHA-1 and HMAC SHA-2 in the Hamming weight model. In: Obaidat MS, Holzinger A, Filipe J (Eds.), E-Business and Telecommunications. Springer, Cham, p.363-379. https://doi.org/10.1007/978-3-319-25915-4_19

Bellare M, Canetti R, Krawczyk H, 1996. Keying hash functions for message authentication. Int Cryptology Conf on Advances in Cryptology, p.1-15. https://doi.org/10.1007/3-540-68697-5_1

Brier E, Clavier C, Olivier F, 2004. Correlation power analysis with a leakage model. In: Joye M, Quisquater JJ (Eds.), Cryptographic Hardware and Embedded Systems. Springer Berlin Heidelberg, p.16-29. https://doi.org/10.1007/978-3-540-28632-5_2

Ding DW, Gao XW, 2012. Design and implementation of SM3 algorithm on FPGA. *Microcomp Appl*, 31(5):26-28 (in Chinese). https://doi.org/10.3969/j.issn.1674-7720.2012.05.009

FIPS, 2002. The Keyed-Hash Message Authentication Code (HMAC). Federal Information Processing Standards Publication, Gaithersburg, MD, USA.

Guo LM, Wang LH, Liu D, et al., 2015. A chosen-plaintext differential power analysis attack on HMAC-SM3. 11th Int Conf on Computational Intelligence and Security, p.350-353. https://doi.org/10.1109/CIS.2015.91

Kocher P, Jaffe J, Jun B, 1999. Differential power analysis. In: Wiener M (Ed.), Advances in Cryptology. Springer Berlin Heidelberg, p.388-397. https://doi.org/10.1007/3-540-48405-1_25

Liu ZB, Ma Y, Jing JW, et al., 2011. Implementation of SM3 HASH function on FPGA. *Netinfo Secur*, 9:191-193, 218 (in Chinese). https://doi.org/10.3969/j.issn.1671-1122.2011.09.059

Ma Y, Xia LN, Lin JQ, et al., 2012. Hardware performance optimization and evaluation of SM3 hash algorithm on FPGA. 14th Int Cryptology Conf on Information and Communications Security, p.105-118. https://doi.org/10.1007/978-3-642-34129-8_10

McEvoy R, Tunstall M, Murphy CC, et al., 2007. Differential power analysis of HMAC based on SHA-2, and countermeasures. 8th Int Conf on Information Security Applications, p.317-332. https://doi.org/10.1007/978-3-540-77535-5_23

Menezes A, van Oorschot PC, Vanstone S, 1996. Hash functions and data integrity. In: Handbook of Applied Cryptography. CRC Press, Boca Raton, USA, p.321-376.

Moradi A, Barenghi A, Kasper T, et al., 2011. On the vulnerability of FPGA bitstream encryption against power analysis attacks: extracting keys from Xilinx Virtex-II FPGAs. Proc 18th ACM Conf on Computer and Communications Security, p.111-124. https://doi.org/10.1145/2046707.2046722

Qu KG, An W, Wu LJ, et al., 2015. A novel masking scheme for SM3 based MAC. *China Commun*, 12(6):12-21. https://doi.org/10.1109/CC.2015.7122475

SCA, 2010. SM3 Cryptographic Hash Algorithm. State Cryptography Administration of China (in Chinese).

Sun W, Liu JR, Gu DW, et al., 2015. Research on power analysis against software-based and hardware-based cryptographic circuits. Int Conf on Computer Science and Communication Engineering, p.1-8.

Tunstall M, Hanley N, McEvoy RP, et al., 2007. Correlation power analysis of large word sizes. IET Irish Signals and Systems Conf, p.13-14.

Wang XY, Yang XW, 2012. Optimization design and implementation of SM3 algorithm based on FPGA. *Comput Eng*, 38(6):244-246 (in Chinese). https://doi.org/10.3969/j.issn.1000-3428.2012.06.081