A Robust Scan-based Side-channel Attack Method against HMAC-SHA-256 Circuits

Daisuke Oku, Masao Yanagisawa, Nozomu Togawa
Dept. of Computer Science and Communications Engineering,
Waseda University
Email: {daisuke.oku, togawa}@togawa.cs.waseda.ac.jp

***Abstract*—A scan-based side-channel attack is still a real threat against a crypto circuit as well as a hash generator circuit, which can restore secret information by exploiting the scan data obtained from scan chains inside the chip during its processing. In this paper, we propose a scan-based attack method against a hash generator circuit called HMAC-SHA-256. Our proposed method restores the secret information by finding out the correspondence between the scan data obtained from a scan chain and the internal registers in the target HMAC-SHA-256 circuit, even if the scan chain includes registers other than the target hash generator circuit and an attacker does not know well the hash generation timing. Experimental results show that our proposed method successfully restores two secret keys of the HMAC-SHA-256 circuit in at most 6 hours.**

***Index Terms*—HMAC, SHA-256, side-channel attack, scan-based attack**

I. Introduction

In recent years, integrated circuits (ICs) are applied to very wide applications, such as in credit cards, medical devices, and others, which often deal with personal and confidential informations. Once these informations leak out, it could cause a serious social risk.

A *side-channel attack* is one of the security attacks that can restore the confidential information inside an LSI (Large Scale Integrated-circuit) chip by observing physical information obtained from the running circuit. A timing attack [1], a fault analysis attack [2], [3], a cache attack [4], and a differential power attack [5]–[8] have been reported so far.

A scan chain is one of the powerful test tools in LSI design, which can control and observe internal registers easily and directly by connecting registers serially in an LSI chip. However, a *scan-based attack*, one of the side-channel attacks, has been reported which can restore the confidential information using the scan data obtained from the scan chain embedded on the target cipher chip [9]–[18]. In [19]–[21], it has pointed out that there still exist LSI chips on which a scan chain can be accessed non-securely. The architectures that can compress and protect scan data and also several secure scan architectures have been proposed [21]. These architectures may have the drawbacks from the viewpoints of area overheads and testability and hence not all the LSI chips always have these secure architectures. This means that still an attacker can observe and access a scan chain in a practical LSI chip and a scan-based attack can be a real threat against a crypto circuit as well as a hash generator circuit.

HMAC is one of the message authentication algorithms [22]. In HMAC, a message and a secret key are input to a hash generator circuit and its corresponding hash value is generated for authentication. Differential power analysis based side-channel attack against HMAC has been proposed in [6]–[8] but there are almost no researches on scan-based attacks against HMAC proposed so far.

In this paper, we propose a scan-based attack method against a hash generator circuit called HMAC-SHA-256. HMAC-SHA-256 is one of the message authentication algorithms using SHA-256 hash function, which is very widely used as a hash generator. Our proposed method finds out the correspondence between the scan data and the internal registers in the target HMAC-SHA-256 circuit by analyzing the scan data obtained from a scan chain and restores the secret information using these data. The method is composed of the following steps: Firstly, we isolate 64 bit-transition groups from scan data by identifying the initial position of each SHA-256 process; After that, we classify 64 bit-transitions groups into 32 pairs; Lastly, we determine whether each bit-transitions group in a pair corresponds to either of the internal register a or e; Experimental results show that our method successfully restores two secret keys from the scan data, even if the scan chain includes other registers than those of HMAC-SHA-256 circuit and we do not well know the SHA-256 process timing in HMAC-SHA-256.

The contributions of the paper are summarized as follows: (1) the proposed scan-based attack method against HMAC-SHA-256 circuits can successfully restore the secret keys, even if the scan chain includes other registers than those of the target HMAC-SHA-256 circuit and we do not well know the SHA-256 process timing; (2) experimental results demonstrate that we can restore the secret keys in a practical amount of analyzing time.

II. HMAC-SHA-256

In this section, we introduce the algorithms of HMAC and SHA-256 and the target HMAC-SHA-256 circuit. The operators used in this paper are summarized in Table I.

A. HMAC [22]

HMAC (Hash-based Message Authentication Code) is one of the widely used message authentication codes, which can be used with any iterative cryptographic hash function. In HMAC,

TABLE I
SUMMARY OF NOTATION IN THIS PAPER.

$\oplus$	Bitwise XOR
$\wedge$	Bitwise AND
$\bar{x}$	Bitwise NOT of x
$\leftarrow$	Substitution
$\boxplus$	$(\mod 2^{32})$-addition
$\|\|$	Concatenation operator
R^n	Right shift by n bits
S^n	Right rotation by n bits

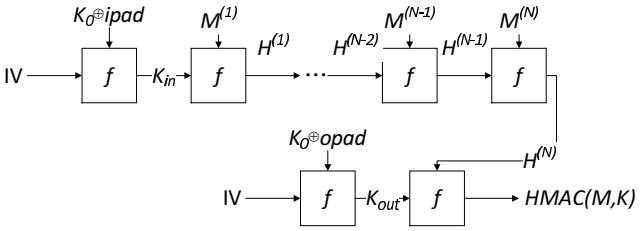

Fig. 1. The block diagram of HMAC.

given an input message M and an original secret key K, an authentication code *HMAC(M,K)* is generated according to the diagram as shown in Fig. 1.

In Fig. 1, f shows a compression function, $ipad$ and $opad$ are constants, and IV is a given initial value. The input message M is partitioned into $M^{(1)}, M^{(2)}, \ldots, M^{(N)}$, each of which has B-bit length. A B-bit value K_0 is generated by the original secret key K as follows:

$$K_0 = \begin{cases} K & (L(K) = B) \\ K \;||\; \underbrace{0 \ldots 0}_{B - L(K)} & (L(K) < B) \\ H(K) \;||\; \underbrace{0 \ldots 0}_{B - L(H(K))} & (B < L(K)) \end{cases}$$

where $L(K)$ shows the bit length of K and $H(K)$ shows the hash code when K is input. Note that $B < L(H(K))$. K_{in}, K_{out} and $H^{(i)}$ $(1 \le i \le N)$ in Fig. 1 are the intermediate hash values generated by the compression function f.

B. SHA-256 [23], [24]

SHA-256 is a cryptographic hash function standardized by NIST which outputs a 256-bit hash value. In SHA-256, an input message is partitioned into 512-bit message blocks $M^{(1)}, M^{(2)}, \ldots, M^{(N)}$,where N shows the number of message blocks. When we partition a message into 512-bit message blocks, the size of the last message block, $M^{(N)}$, may become smaller than 512 bits. In SHA-256, '1', a number of '0's, and a 64-bit integer indicating the entire message size are appended to the end of the last message block. If the last message block is too long to accommodate all the above padding bits, the last two message blocks are used.

The algorithm of SHA-256 is shown in Algorithm 1. The algorithm is applied to every message block $M^{(i)}$ $(1 \le i \le N)$ repeatedly and the main loop for $M^{(i)}$ is composed of the three phases: In PHASE 1, the internal registers are initialized. Let $H^{(i-1)}$ be a 256-bit intermediate hash value generated for the previous message block $M^{(i-1)}$. We assume $H^{(0)} = IV$. $H^{(i-1)}$ is partitioned into eight 32-bit sub-hash values, $H_1^{(i-1)}, \ldots, H_8^{(i-1)}$. Let a, b, c, d, e, f, g, h be the 32-bit internal registers in SHA-256 circuit. Then these registers are initialized using the $(i-1)$-th intermediate hash value $H^{(i-1)}$.

Algorithm 1 SHA-256

for $i = 1$ to N **do**
  {**PHASE 1:** Initialize internal registers}
  $a \leftarrow H_1^{(i-1)}$; $b \leftarrow H_2^{(i-1)}$; $c \leftarrow H_3^{(i-1)}$; $d \leftarrow H_4^{(i-1)}$;
  $e \leftarrow H_5^{(i-1)}$; $f \leftarrow H_6^{(i-1)}$; $g \leftarrow H_7^{(i-1)}$; $h \leftarrow H_8^{(i-1)}$;
  {**PHASE 2:** Apply compression function}
  for $j = 0$ to 63 **do**
    $T_1 \leftarrow h \boxplus Rot_2(e) \boxplus Ch(e,f,g) \boxplus K_j \boxplus W_j$;
    $T_2 \leftarrow Rot_1(a) \boxplus Maj(a,b,c)$;
    $h \leftarrow g$; $g \leftarrow f$; $f \leftarrow e$;
    $e \leftarrow d \boxplus T_1$;
    $d \leftarrow c$; $c \leftarrow b$; $b \leftarrow a$;
    $a \leftarrow T_1 \boxplus T_2$;
  end for
  {**PHASE 3:** Calculate i-th intermediate hash value $H^{(i)}$}
  $H_1^{(i)} \leftarrow a \boxplus H_1^{(i-1)}$; $H_2^{(i)} \leftarrow b \boxplus H_2^{(i-1)}$;
  $H_3^{(i)} \leftarrow c \boxplus H_3^{(i-1)}$; $H_4^{(i)} \leftarrow d \boxplus H_4^{(i-1)}$;
  $H_5^{(i)} \leftarrow e \boxplus H_5^{(i-1)}$; $H_6^{(i)} \leftarrow f \boxplus H_6^{(i-1)}$;
  $H_7^{(i)} \leftarrow g \boxplus H_7^{(i-1)}$; $H_8^{(i)} \leftarrow h \boxplus H_8^{(i-1)}$;
end for
$H^{(N)} = (H_1^{(N)} || H_2^{(N)} || \cdots || H_8^{(N)})$;

In PHASE 2, the compression function is applied to update the internal registers as in Algorithm 1, where $Ch(e,f,g), Maj(a,b,c), Rot_1$ and Rot_2 are calculated by:

$$Ch(e,f,g) = (e \wedge f) \oplus (\bar{e} \wedge g) \quad (1)$$

$$Maj(a,b,c) = (a \wedge b) \oplus (a \wedge c) \oplus (b \wedge c) \quad (2)$$

$$Rot_1(a) = S^2(a) \oplus S^{13}(a) \oplus S^{22}(a) \quad (3)$$

$$Rot_2(e) = S^6(e) \oplus S^{11}(e) \oplus S^{25}(e) \quad (4)$$

In PHASE 2, K_j $(j = 0 \ldots 63)$ shows the 32-bit constant defined by the specification of SHA-256. The 512-bit message block $M^{(i)}$ is further partitioned into a set of 32-bit values $M_0^{(i)}, \ldots, M_{15}^{(i)}$. Then W_j in PHASE 2 is given by:

$$W_j = \begin{cases} M_j^{(i)} & (j = 0, 1, \ldots, 15) \\ X_j & (j = 16, 17, \ldots, 63) \end{cases} \quad (5)$$

where X_j is defined by:

$$X_j = \sigma_1(W_{j-2}) \boxplus W_{j-7} \boxplus \sigma_0(W_{j-15}) \boxplus W_{j-16} \quad (6)$$

In the above equation, $\sigma_0(x) = S^7(x) \oplus S^{18}(x) \oplus R^3(x)$ and $\sigma_1(x) = S^{17}(x) \oplus S^{19}(x) \oplus R^{10}(x)$.

In PHASE 3, the i-th intermediate hash value $H^{(i)}$ for $M^{(i)}$

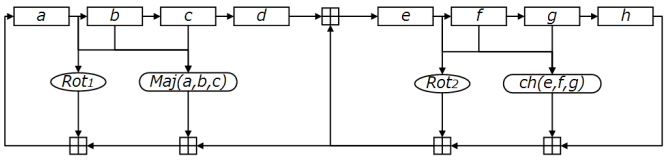

Fig. 2. The block diagram of the compression function in PHASE 2 of Algorithm 1.

is finally calculated as in Algorithm 1.

The block diagram of the compression function and message schedule used in SHA-256 is shown in Fig. 2.

C. HMAC-SHA-256

In HMAC-SHA-256, the compression function given by SHA-256 is used in each of the f functions in HMAC as in Fig. 1. HMAC-SHA-256 is used in IPsec and SSL/TLS [25], [26], for example.

In HMAC-SHA-256, a 512-bit secret key is used and an input message M is partitioned into N 512-bit message blocks, $M^{(1)}, M^{(2)}, \ldots, M^{(N)}$. The SHA-256 circuit operates $(N+3)$ times as depicted in Fig. 2. However, the SHA-256 circuit does not necessarily operate continuously, since the inputs to every SHA-256 process must be prepared beforehand and some amount of calculation is necessary before the SHA-256 process begins. Furthermore, not all the inputs to the SHA-256 process are the same as in Fig. 2 and thus some of their preprocesses may be different from the others.

In this paper, we employ the HMAC-SHA-256 circuit as a target hash generator circuit, which includes a single SHA-256 circuit block inside. The SHA-256 circuit is repeatedly used to calculate the compression function, which performs PHASE 1 in one clock cycle, PHASE 2 in 64 clock cycles and PHASE 3 in one clock cycle in Algorithm 1, i.e, The SHA-256 circuit performs Algorithm 1 in 66 clock cycles. Since the SHA-256 circuit operates $(N+3)$ times in total, then the HMAC-SHA-256 process requires at least $(N+3)\times 66$ clock cycles. However, we have to require several more clock cycles to prepare the inputs to every SHA-256 process and hence we require more than $(N+3)\times 66$ clock cycles to complete the HMAC-SHA-256 process.

For example, Fig. 3 shows the scan data obtained while running the target HMAC-SHA-256 circuit. In this figure, the gray areas show the extra clock cycles and thus they show *extra scan data* to prepare the inputs to every SHA-256 process.

III. Assumptions

In [9], [10], it is assumed that only the register of the target encryption circuit is connected to the scan chain. However, even a single LSI chip has control circuits, memory blocks, and others inside, in addition to the target encryption circuit, and all the registers of these circuits may be connected to its scan chain. In this paper, we assume a full scan design where

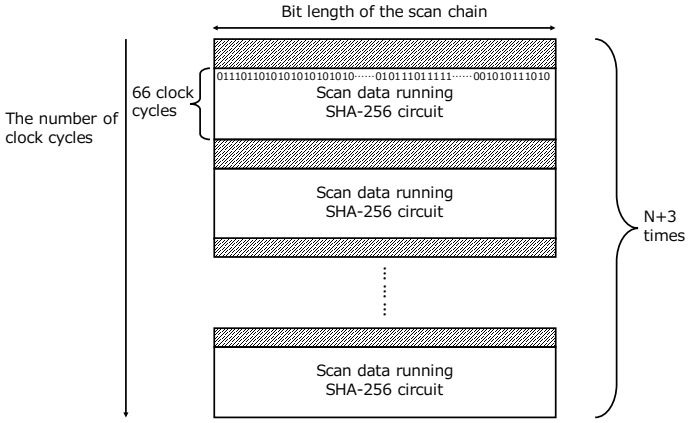

Fig. 3. The scan data obtained from the assumed HMAC-SHA-256 circuit.

all the registers in the LSI chip including the target HMAC-SHA-256 circuit and other circuit blocks are connected to the scan chain.

As in [16], [18], we assume the following assumptions:

1) Attackers know that the full scan design is embedded in LSI and the scan chain does not change dynamically and the scan data is not compressed.
2) Attackers can input any message to the target HMAC-SHA-256 circuit and obtain the scan data from the scan chain at any time.
3) Attackers do not know the connection order of the registers in the scan chain of the target HMAC-SHA-256 circuit nor the number and type of registers in the scan chain of the target HMAC-SHA-256 circuit. Thus the scan data itself is meaningless to attacker as it is.

K_{in} and K_{out} in Fig. 1 are considered to be the secret keys in HMAC-SHA-256 [6]–[8] and thus we restore K_{in} and K_{out} using the scan data based on the assumptions above.

IV. Scan-based Attack against HMAC-SHA-256

In this section, we propose a scan-based attack against HMAC-SHA-256 circuit using the scan data obtained from its scan chain. HMAC-SHA-256 circuit does not generate a hash value continuously. Our algorithmis composed of four steps below:

A. Isolate 64 bit-transition groups from scan data (STEP 1)

When we look at the SHA-256 compression function of Algorithm 1, the value of the register a moves to the registers b, c and d in this order as in Fig. 2. The value of the register e also moves to the register f, g and h in this order. Let x_i denote the i-th bit of register x. Then a_i moves to b_i, c_i and d_i $(0 \le i \le 31)$ in this order. Similarly, e_i moves to f_i, g_i and h_i in this order. A *bit-transition group* refers to the bit positions of a_i, b_i, c_i and d_i or those of e_i, f_i, g_i and h_i. We have totally 64 bit-transition groups since every internal register is 32-bit long and a bit-transition group initiates either from the register a or from the register e.

Now we find out these 64 bit-transition groups. We first vertically arrange the scan data obtained from a scan chain

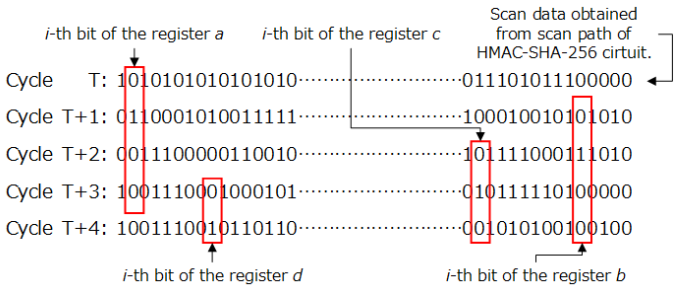

Fig. 4. Bit-transitions of a_i in the scan data.

at different clock cycles and focus on a particular vertical bit sequence as in Fig. 4. In Fig. 4, the scan data at Cycle T to Cycle $(T+4)$ are shown. Every vertical bit sequence shows a particular bit change at different clock cycles and called a *scan signature*.

Assume that the scan signature of a_i is given by the leftmost rectangle of Fig. 4. Since a_i is moved to b_i at the next clock cycle, a_i value at Cycle T must be moved to b_i value at Cycle $(T+1)$. In the same way, a_i value at Cycle $(T+1)$ must be moved to b_i value at Cycle $(T+2)$. This means that the scan signature of a_i from Cycle T to Cycle $(T+n)$ must be moved to the scan signature of b_i from Cycle $(T+1)$ to Cycle $(T+n+1)$. In the same way, the scan signature of b_i from Cycle $(T+1)$ to Cycle $(T+n+1)$ must be moved to the scan signature of c_i from Cycle $(T+2)$ to Cycle $(T+2+n)$. If n is large enough, or we input many messages and obtain scan data, we can find out at the same bit positions the identical scan signature of $(n+1)$-bit length in the scan data whose starting clock cycle is different by one bit from each other. We can finally isolate the 64 bit-transition groups in the scan data, even though we cannot know every bit-transition group corresponds to any one bit of internal registers.

B. Identify the initial position of SHA-256 process from the scan data (STEP 2)

As discussed in Section II-C, we assume here that the scan data includes the extra data for preprocessing which do not directly correspond to SHA-256 processes (the gray areas as shown in Fig 3). STEP 1 above can be applied only when the scan signatures do not include the extra scan data as shown in the bold long rectangles in Fig. 5 and it cannot be directly applied if the scan signatures include the extra scan data as shown in the dotted long rectangles in Fig. 5. We have to find which parts of the obtained scan data correspond to the SHA-256 processes, i.e., we have to effectively eliminate the extra scan data, so that STEP 1 above can be applied to them.

Firstly, the entire scan data is partitioned into several 32 clock cycle scan data blocks as shown in Fig. 5 from the top of the scan data. Then some of the partitioned scan data blocks always contain no extra scan data for preprocessing, since the SHA-256 circuit requires 66 clocks for a single SHA-256 process. In this case, STEP 1 above can successfully isolate the bit-transition groups.

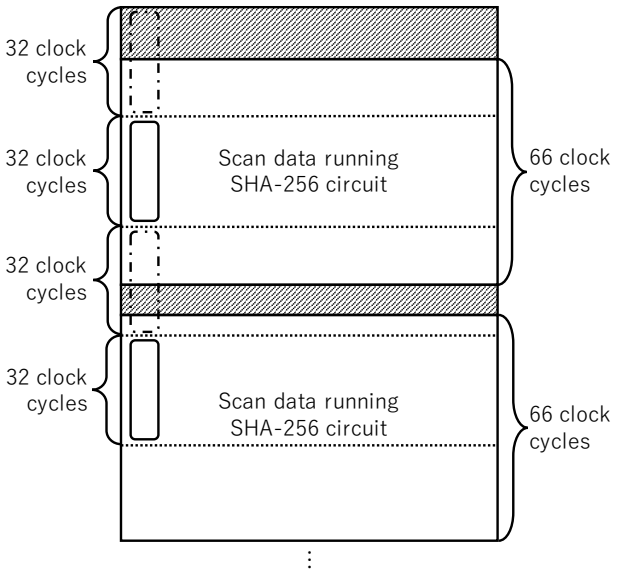

Fig. 5. Search within scan data including random data.

Once we can isolate the bit-transition groups in some of the scan data blocks, the initial positions at which the SHA-256 circuit starts its process are identified by going back to the previous clock cycles from the clock cycles where the transition group can be successfully isolated and then we can identify the scan data which directly correspond to SHA-256 processes. Since we know that every SHA-256 process operates in 66 clock cycles, we can distinguish between white scan data and gray scan data as in Fig. 5.

By combining STEP 1 and STEP 2, we can successfully obtain initial positions and bit-transition groups.

C. Classify 64 bit-transition groups into 32 pairs (STEP 3)

In STEP 1 and STEP 2, the 64 bit-transition groups are isolated from the scan data. After that, in STEP 3, these 64 bit-transition groups and the scan data for which the internal position can be identified are classified into 32 pairs $P_0, \ldots, P_{31}$ of bit-transition groups, i.e., each pair P_i $(0 \leq i \leq 31)$ has two bit-transition groups and each bit-transition group in P_i corresponds to the i-th bit in the internal register a or e.

First we prepare two messages m^1 and m^2 and consider that the first message block from m^1 is input to the target circuit at Cycle $(T+1)$. Assume that the registers a to d and e to h are initialized at PHASE 1 of Algorithm 1 at Cycle T. Then the registers a and e are updated at Cycle $(T+1)$ according to the PHASE 2 of Algorithm 1 using the input message m^1. This update at Cycle $(T+1)$ is described by:

$$a^1 = X \boxplus W_0^1 \quad (7)$$

$$e^1 = Y \boxplus W_0^1 \quad (8)$$

where a^1 and e^1 are the updated values for a and e and X and Y are the constants calculated using the initial values of the registers a to d and e to h. W_0^1 is the first 32 bits of the input message m^1 as in Eqn. (5).

TABLE II
EXAMPLE OF ASCII SYMBOLS WITH HAMMING DISTANCE 1.

A pair of ASCII symbols	Bit difference
N/A	10000000
(q, 1)	01000000
(q, Q)	00100000
(q, a)	00010000
(q, y)	00001000
(q, u)	00000100
(q, s)	00000010
(q, p)	00000001

In the same way, when the first message block from the message m^2 is input to the target circuit at time $(T+1)$, the updated values of the registers a^2 and e^2 are described by:

$$a^2 = X \boxplus W_0^2 \quad (9)$$
$$e^2 = Y \boxplus W_0^2 \quad (10)$$

where W_0^2 is the first 32 bits of the input message m^2. Since the initial values of the registers a to h here are completely the same as those of Eqns. (7) and (8), we can use the same X and Y values in Eqns. (9) and (10).

Now we prepare a pair of messages, m^1 and m^2, so that $W_0^2 = W_0^1 + \alpha$, where the MSB of α is one and the other bits are zero. At that time, the MSBs of a^2 and e^2 must be inverted from those of a^1 and e^1 and the other bits are completely the same. Then, by observing the bit difference between the bit-transition groups obtained by inputting m^1 and m^2, we can find out the two bit-transition groups which correspond to the MSBs of the internal registers a and e.

However, according to the specification of HMAC-SHA-256, its input massage must be represented by a sequence of 8-bit ASCII codes and we cannot have two ASCII codes whose difference is MSB only. Table II shows an example pair of ASCII symbols and its bit difference. Now the *first MSB* refers to MSB itself and the *second MSB* refers to the second bit from MSB in each register. In order to determine the two bit-transition groups corresponding to the first MSBs of a and e, we give "qqqq" and "1qqq" to W_0^1 and W_0^2, for example. As a result, according to Eqns. (7)–(10), the second MSBs must be inverted in a^2 and e^2 compared to a^1 and e^1, and the first MSBs may also be inverted in a^2 and e^2 if there exist carry bits from the second MSBs to the first MSBs in calculating Eqns. (7)–(10). In other words, the first and second MSBs may be all inverted (totally 4 bits) in a^2 and e^2 at most and only the two second MSBs are inverted (totally 2 bits) in a^2 and e^2 at least.

When we prepare many such pairs of input messages and count the inverted bits at each case, we can know which bits are the first MSBs and which bits are the second MSBs of the registers a and e, since the second MSBs of the registers a and e are always inverted while the first MSBs of the registers a and e are inverted in some cases. Once we can find out the two bit-transition groups which correspond to the first MSBs of the internal registers a and e as well as the two bit-transition groups which correspond to their second MSBs, we can find out the other bit correspondence from MSB to LSB in the same way. We finally have 32 pairs, $P_0, \ldots, P_{31}$, of bit-transition groups. Note that, we cannot distinguish in each pair, P_i, which one corresponds to the register a and which one corresponds to the register e at this time.

D. Determine whether each bit-transition group in a pair corresponds to register a or register e (STEP 4)

Finally, in STEP 4, we determine whether each bit-transition group in P_i corresponds to the register a or register e.

According to PHASE 2 of Algorithm 1, the register values at Cycle T and those at Cycle $(T+1)$ must satisfy:

$$a^{T+1} \boxplus d^T = e^{T+1} \boxplus Rot_1(a^T) \boxplus Maj(a^T, b^T, c^T). \quad (11)$$

Since the above equation can be calculated in a bit-wise manner, we determine from LSB to MSB the final bit positions of each register in the 64 bit-transition groups so that they satisfy Eqn. (11). Overall, we can know bit-correspondence between the scan data and the bit positions of internal registers.

Since the secret key K_{in} is used at the first process of SHA-256 circuit and the secret key K_{out} is used at the last process of SHA-256 as shown in Fig. 1, we can easily know their input timing and thus we can restore the K_{in} and K_{out} values using the scan data.

V. EXPERIMENTAL EVALUATIONS

In this section, we demonstrate the experimental results to restore the secret keys K_{in} and K_{out} in HMAC-SHA-256 circuit by using our proposed method. Our proposed method is implemented in python on the computer environment, Intel Core i5 (2.9GHz) with 8GB main memory.

A. Setup

In this experiment, we make sure whether the secret keys K_{in} and K_{out} can be restored when the length of the scan chain is changed. The scan chain length of the internal registers a to h becomes $32 \times 8 = 256$ bits. When we change its length, we add random bits to it. These extra bits inserted randomly into the 256-bit original scan chain can be considered to be the scan data for the circuit blocks other than the target HMAC-SHA-256 circuit. Then we have the scan chain lengths from 256 bits to 2048 bits. Furthermore, 2 to 100 clock cycle random scan data were inserted as extra scan data for preprocessing to every SHA-256 process. For each scan chain length, 50 types of secret keys were randomly prepared and given to the target HMAC-SHA-256 circuit.

B. Results

In all the cases, our proposed method successfully restores the secret keys K_{in} and K_{out} and their CPU times are summarized in Fig. 6. At that time, we have prepared at most 150 messages in total. As Fig. 6 demonstrates, the maximum time, the minimum time, and the average time to analyze the two secret keys in HMAC-SHA-256 are shown for each bit

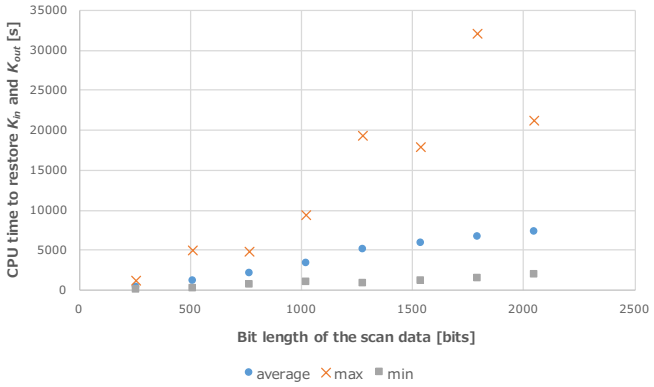

Fig. 6. The CPU time to restore the secret keys K_{in} and K_{out}.

length of the scan chain. We take around 6 hours to restore the secret keys in HMAC-SHA-256 circuit, even if the scan chain length becomes 2048 bits and include extra scan data for preprocessing.

Once K_{in} and K_{out} are restored, an attacker can replicate the HMAC-SHA-256 circuit which can generate a correct hash value from any input message and he/she can establish a secure communication informally. The experimental result here demonstrates that a scan-based attack against HMAC-SHA-256 is definitely a real threat.

VI. Conclusions

In this paper, we have proposed a scan-based attack method against HMAC-SHA-256 circuit. Our proposed method can restore the secret key used for the authentication by finding out the correspondence between the scan data obtained from a scan chain and the internal registers in HMAC-SHA-256 circuit, even if the scan chain includes other registers than internal registers in the target HMAC-SHA-256 circuit. Experimental results show that our method successfully restores the secret keys from the scan data in at most 6 hours to restore the secret keys in HMAC-SHA-256 circuit, even if the scan chain length becomes 2048 bits.

In the future, we will apply our scan-based attacks to other cipher circuits and evaluate if they are a real threat or not. Furthermore, we will develop a protection method against scan-based attacks.

Acknowledgment

This research and development work was supported in part by the MIC/SCOPE #171503005.

References

[1] P. C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems, " *Lecture Notes in Computer Science* vol. 1109, pp. 104–113, 1996.

[2] E. Biham and A. Shamir, "Differential fault analysis of secret key cryptosystems," *Lecture Notes in Computer Science*, vol. 1294, pp. 513–525, 1997.

[3] D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the importance of checking cryptographic protocols for faults," *Lecture Notes in Computer Science*, vol. 1233, pp. 37–51, 1997.

[4] Y. Tsunoo, E. Tsujihara, K. Minematsu, and H. Miyauchi, "Cryptanalysis of block ciphers implemented on computers with cache," *Lecture Notes in Computer Science*, vol. 2779, pp. 62–76, 2003.

[5] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Proc. CRYPTO '99*, Springer-Verlag, pp. 388–397, 1999.

[6] S. Belaïd, L. Bettale, E. Dottax, L. Genelle, and F.Rondepierre, "Differential power analysis of HMAC SHA-2 in the hamming weight model," in *Proc. International Conference on Security and Cryptography (SECRYPT 2013)*, pp.230–241, 2013.

[7] R. McEvoy, M. Tunstall, C. C. Murphy, and W. P. Marnane, "Differential power analysis of HMAC based on SHA-2, and countermeasures," *Lecture Notes in Computer Science,* vol. 4867, pp. 317–332, 2007.

[8] K. Okeya, "Side channel attacks against HMACs based on block cipher based hash functions," *Lecture Notes in Computer Science*, vol. 4058, pp. 432–443, 2006.

[9] B. Yang, K. Wu and R. Karriv "Scan based side channel attack on dedicated hardware implementations of Data Encryption Standard," in *Proc. of International Test Conference*, pp. 339–344, 2004.

[10] B. Yang, K. Wu and R. Karri, "Secure scan: a design-for-test architecture for crypto chips," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 10, pp. 2287–2293, 2006.

[11] R. Nara, N. Togawa, M. Yanagisawa, T.Ohtsuki, "A scan-based attack based on discriminators for AES cryptosystems," *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E92-A. no. 12, pp. 3229–3237, 2009.

[12] J. D. Rolt, G. D. Natale, M. Flottes, and B. Rouzeyre, "A novel differential scan attack on advanced DFT structures," ACM Transactions on Design Automation of Electronic Systems (TODAES), vol. 18, no. 4, pp. 58:1–58:22, 2013.

[13] S. S. Ali, S. Saeed, O. Sinanoglu, and R. Karri, "Novel test-mode-only scan attack and countermeasure for vompression-based scan architebures," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 5, pp. 808–821, 2015.

[14] R. Nara, K. Satoh, M. Yanagisawa, T. Ohtsuki, and N. Togawa, "Scan-based side-channel attack against RSA cryptosystems using scan signatures," *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E93-A, no. 12, pp. 2481–2489, Dec. 2010.

[15] R. Nara, M. Yanagisawa, T. Ohtsuki and N. Togawa, "Scan vulnerability in elliptic curve cryptosystems," *IPSJ Trans. System LSI Design Methodology*, vol. 4, February issue, pp. 47–59, Feb. 2011.

[16] M. Agrawal, S. Karmakar, D. Saha, and D. Mukhopadhyay, "Scan based side channel attacks on stream ciphers and their counter-measures," *Lecture Notes in Computer Science*, vol. 5365, pp. 226–238, 2008.

[17] Y. Liu, K. Wu, and R. Karri, "Scan-based attacks on linear feedback shift register based stream ciphers," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 16, no. 2, pp. 20:1–20:15, 2011.

[18] M. Fujishiro, M. Yanagisawa, and N. Togawa, "Scan-based attack against Trivium stream cipher using scan signatures," *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, vol.E97-A no.7 pp.1444–1451, 2014.

[19] A. Das, J. Da Rolt, S.Ghosh, S. Seys, S. Dupuis, G. Di Natale, M. Flottes, B. Rouzeyre, and I. Verbauwhede, "Secure JTAG Implementation Using Schnorr Protocol," *Journal of Electronic Testing,* vol. 29, no. 2, pp. 193–209, Apr. 2013.

[20] E. DeBusschere, and M. McCambridge, "Modern Game Console Exploitation," Technical Report, Department of Computer Science, University of Arizona, 2012.

[21] J. Da Rolt, A. Das, G. Di Natale, M. Flottes, B. Rouzeyre, and I. Verbauwhede, "Test Versus Security: Past and Present," *IEEE Trans. Emerging Topics in Computing*, vol. 2, no. 1, pp. 50–62, 2014.

[22] "FIPS 198-1 The Keyed-Hash Message Authentication Code," http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.198-1.pdf.

[23] "Descriptions of SHA-256, SHA-384, and SHA-512," http://www.iwar.org.uk/comsec/resources/cipher/sha256-384-512.pdf.

[24] "FIPS 180-4 Secure Hash Standard," http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf.

[25] T. Dierks, and E. Rescorla, "Request for Comments 5246: The Transport Layer Security (TLS) protocol version 1.2," IETF, https://tools.ietf.org/html/rfc5246, August 2008.

[26] S. Kelly, and S. Frankel, "Request for Comments 4868: Using HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512 with IPsec," IETF, https://tools.ietf.org/html/rfc4868, May 2007.