Zhang C, Liu JR, Gu DW *et al.* Side-channel analysis for the authentication protocols of CDMA cellular networks. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 34(5): 1079–1095 Sept. 2019. DOI 10.1007/s11390-019-1961-5

Side-Channel Analysis for the Authentication Protocols of CDMA Cellular Networks

Chi Zhang[1], Jun-Rong Liu[1,2], Da-Wu Gu[1,*], *Distinguished Member, CCF, Member, ACM*, Wei-Jia Wang[3] Xiang-Jun Lu[1], Zheng Guo[1,2], and Hai-Ning Lu[1,4]

[1] *School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai 200240, China*

[2] *ZhiXun Crypto Testing and Evaluation Technology Co., Ltd., Shanghai 200240, China*

[3] *Crypto Group, Electrical Engineering Department, Institute of Information and Communication Technologies Catholic University of Louvain, Louvain-la-Neuve B-1348, Belgium*

[4] *Shanghai Viewsource Information Science and Technology Co., Ltd, Shanghai 200240, China*

E-mail: zcsjtu@sjtu.edu.cn; liujunrong@zxcsec.com; dwgu@sjtu.edu.cn; weijia.wang@uclouvain.be luxiangjun@sjtu.edu.cn; guozheng@zxcsec.com; haining.lu@viewsources.com

Received October 25, 2018; revised July 28, 2019.

Abstract Time-division multiple access (TDMA) and code-division multiple access (CDMA) are two technologies used in digital cellular networks. The authentication protocols of TDMA networks have been proven to be vulnerable to side-channel analysis (SCA), giving rise to a series of powerful SCA-based attacks against unprotected subscriber identity module (SIM) cards. CDMA networks have two authentication protocols, cellular authentication and voice encryption (CAVE) based authentication protocol and authentication and key agreement (AKA) based authentication protocol, which are used in different phases of the networks. However, there has been no SCA attack for these two protocols so far. In this paper, in order to figure out if the authentication protocols of CDMA networks are sufficiently secure against SCA, we investigate the two existing protocols and their cryptographic algorithms. We find the side-channel weaknesses of the two protocols when they are implemented on embedded systems. Based on these weaknesses, we propose specific attack strategies to recover their authentication keys for the two protocols, respectively. We verify our strategies on an 8-bit microcontroller and a real-world SIM card, showing that the authentication keys can be fully recovered within a few minutes with a limited number of power measurements. The successful experiments demonstrate the correctness and the effectiveness of our proposed strategies and prove that the unprotected implementations of the authentication protocols of CDMA networks cannot resist SCA.

Keywords authentication protocol, cellular authentication and voice encryption (CAVE), code-division multiple access (CDMA), secure hash algorithm 1 (SHA-1), side-channel analysis

1 Introduction

A cellular network is a wireless communication network that provides services through a number of base stations called cells, each covering a limited area. Such a network is designed to ensure that users can continue to communicate even if they cross between areas covered by different base stations. In recent years, with the development of the Internet and smartphones, massive amounts of user data have begun to be transmitted over cellular networks. Thus, protecting the privacy of users and ensuring the confidentiality of data in cellular networks are critical. An authentication protocol is a mechanism through which a network provides a secure channel over which to deliver data. Diverse cellular networks exist, and they can be divided into two main groups depending on the technologies on which they are based: time-division multiple access (TDMA) and code-division multiple access (CDMA)[1]. More-

Regular Paper

This work was supported by State Grid Science and Technology Project of China under Grant No. 546816190003.

*Corresponding Author

over, these two branches of cellular technology use different authentication protocols.

In the TDMA branch, the second, the third and the fourth generations of the technology (2G, 3G, and 4G) correspond to the global system for mobile communications (GSM), the universal mobile telecommunications system (UMTS) and the long-term evolution (LTE) standards, respectively[2]. GSM uses a one-way authentication protocol with many potential vulnerabilities, such as a lack of confirmation of the identity of the base station. Its core encryption algorithm is a block cipher called COMP128①. The UMTS and LTE standards utilize a superior authentication framework called authentication and key agreement (AKA)②, which has two officially recommended instances: MILENAGE③ and TUAK④. These instances use the advanced encryption standard (AES) and Keccak[3] as their core encryption functions. In the CDMA branch, cdmaOne and CDMA2000 are the 2G and the 3G standards, respectively[4]. Similar to GSM, the cdmaOne also uses a one-way authentication protocol; however, its cryptographic function is a hash-like function called cellular authentication and voice encryption (CAVE)⑤, which is different from the block cipher used in GSM. We refer to the corresponding authentication protocol as the CAVE-based protocol in this paper. The CDMA2000 uses different authentication protocols corresponding to different development periods. Legacy CDMA2000 1X networks use the CAVE-based protocol, whereas newer CDMA2000 1X and 1xEV-DO networks both use AKA as their authentication framework. Unlike MILENAGE and TUAK, the AKA instance applied in CDMA2000 uses secure hash algorithm 1 (SHA-1) as its core encryption algorithm⑥. The corresponding authentication protocol is called the AKA-based protocol in this paper. These authentication protocols are implemented on a subscriber identity module (SIM) card, which is actually an embedded system. The SIM card stores an authentication key in its memory for generating an authentication vector and uses access control for this secret key. The authentication key is quite sensitive, and its leakage could destroy the whole security framework and lead to a series of problems, e.g., cloning of the SIM card. CDMA2000 networks are widely used all over the world⑦. In addition to public mobile cellular networks, CDMA-based technologies are also used in private networks as part of critical infrastructure. For example, ArgoNET⑧ in Austria and Alliander⑨ in the Netherlands have established private networks used for smart metering based on CDMA2000, launched in November 2014 and May 2016, respectively. These recent applications demonstrate that CDMA networks are still active, and thus their security should not be neglected.

Side-channel analysis (SCA) is a powerful class of attacks on implementations of cryptographic algorithms, especially for embedded systems. The vanilla implementations of authentication protocols in the TDMA branch discussed above have all been proven to be vulnerable to SCA[5], meaning that the authentication key can be fully recovered via specific SCA strategies. Rao *et al.*[6] introduced a class of SCA attacks called partitioning attacks based on an analysis of the power consumption during the execution of COMP128 on an 8-bit SIM card, and they showed that through such an attack, the secret key could be recovered within a few minutes. Zhou *et al.*[7] showed that the implementation of the COMP128 on a 16-bit CPU is also susceptible to SCA. Liu *et al.*[8] showed that the secret key of MILENAGE could be recovered within 10 minutes by means of SCA with a few power traces. Maghrebi and Bringer[9] successfully mounted a side-channel attack on an unprotected implementation of TUAK.

However, to the best of our knowledge, no SCA attack strategy has yet been developed for the authentication protocols of CDMA networks. The aim of this paper is to present SCA attack strategies for the exiting authentication protocols of CDMA networks and to prove that unprotected implementations of these pro-

①Briceno M, Goldberg I, Wagner D. GSM Cloning, Apr. 1998. http://www.isaac.cs.berkeley.edu/isaac/gsm-faq.html, July 2019.

②3G security; security architecture, Mar. 2017. https://www.3gpp.org/ftp/Specs/archive/33_series/33.102/, July 2019.

③Specification of the MILENAGE algorithm set, Mar. 2017. https://www.3gpp.org/ftp/Specs/archive/35_series/35.205/, July 2019.

④Specification of the TUAK algorithm set, Mar. 2017. https://www.3gpp.org/ftp/Specs/archive/35_series/35.231/, July 2019.

⑤Common cryptographic algorithms, May. 2009. https://www.3gpp2.org/Public_html/Specs/S.S0053-0_v2.0.pdf, July 2019.

⑥Enhanced cryptographic algorithms, Jan. 2008. https://www.3gpp2.org/Public_html/Specs/S.S0055-A_v4.0_080118.pdf, July 2019.

⑦CDMA2000 networks list, 2017. https://en.wikipedia.org/wiki/List_of_CDMA2000_networks, July 2019.

⑧ArgoNet, 2014. https://www.argonet.at/de/firma/ueber-uns.html, July 2019.

⑨Alliander, 2016. https://www.alliander.com/nl/media/nieuws/utility-connect-neemt-draadloos-datacommunicatienetwerk-gebruik, July 2019.

tocols are not secure against SCA. Our major contributions are summarized as follows.

• We investigate the existing authentication protocols in CDMA networks, namely the CAVE-based authentication protocol and the AKA-based authentication protocol. We systematically present and analyze the structures of these two protocols and their cryptographic algorithms. We find the side-channel weaknesses of the two protocols when they are implemented on embedded systems. For the CAVE-based protocol, we find that its look-up table operation can be attacked by correlation power analysis (CPA)[10] and its "if" statement is unbalanced, which can be attacked by simple power analysis (SPA)[11]. For the AKA-based protocol, we compile the sample code provided by the specification and check the assembly code. We find that its circular shift operation is a proper target of CPA.

• Based on these weaknesses, we propose the specific attack strategies for the two protocols through which the authentication keys could be extracted easily. The leakage of the authentication key could destroy the whole security framework of the authentication protocol and lead to a series of problems, e.g., cloning of the SIM card.

• We verify our strategies on an 8-bit microcontroller and a real-world SIM card. The experimental results show that the authentication keys can be fully recovered within a few minutes with a limited number of power measurements. This is the first work to report successful key recovery attacks for the authentication protocols of CDMA networks and give the negative conclusion that unprotected implementations of the authentication protocols of CDMA networks cannot resist SCA.

The paper is organized as follows. In Section 2, we present the CAVE-based and the AKA-based protocols and their cryptographic algorithms. In Section 3, we briefly describe the CPA and SPA methods and introduce detailed SCA attack strategies for the two protocols. Then, we report our practical experiments in Section 4. At last, this paper is concluded in Section 5.

2 Background

2.1 CAVE-Based Authentication Protocol

The 2G CDMA standard, cdmaOne, uses a CAVE-based protocol for authentication. The process of this authentication protocol is illustrated in Fig.1. The mobile station (MS) is the device on the user side, such as a mobile phone. Usually, there is a SIM card inside the MS. A 64-bit master key for the protocol, called the authentication key (*A-key*), is stored in the permanent memory of a SIM card at the factory and in

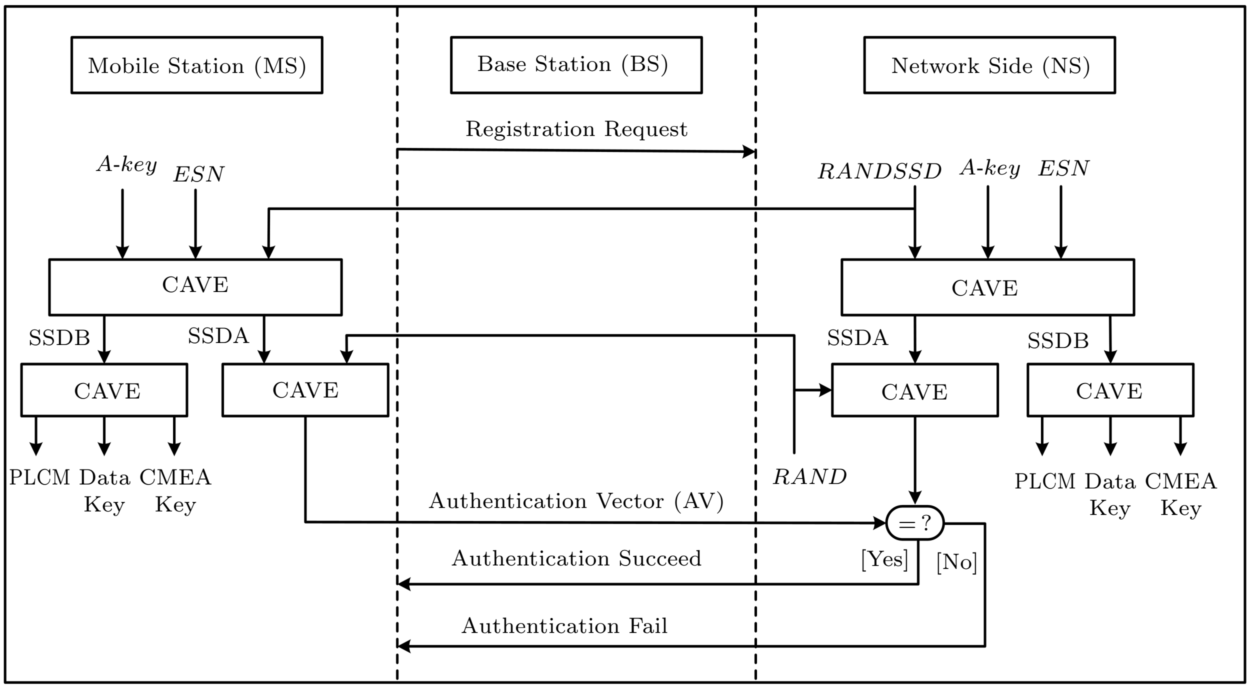


Fig.1. CAVE-based authentication protocol. PLCM (private long code mask), data key, and CMEA Key (cellular message encryption algorithm key) are three derived keys used to support voice, signaling and data encryption respectively.

the database on the network side (NS). It is the secret data known only by the SIM card and the NS; no other party has access to *A-key*. The electronic serial number (*ESN*) is a public 32-bit number that acts as a unique identifier of the MS. When the MS wants to register in the cellular network, it will send a registration request with its *ESN* to the NS through a base station (BS). After receiving the request, the NS will retrieve *A-key* corresponding to the *ESN* and generate a 56-bit random challenge ($RANDSSD$) and send it to the MS. Simultaneously, the MS and the NS run shared secret data (SSD) generation procedure, which is actually a CAVE algorithm with specific inputs composed of *A-key*, *ESN*, RANDSSD, and others. The generated SSD is split into two parts, a 64-bit SSDA and a 64-bit SSDB. Then, the SSDA and another random challenge from the NS are used to generate an 18-bit authentication vector (AV) via the CAVE on both sides. The AV generated by the MS is sent to the NS for authentication. If the AVs on both sides match, the MS will be considered legal. Then, the MS and the NS will use session keys generated via CAVE with the SSDB to encrypt the voice data and any other data to be sent and will communicate through the BS. If the AVs do not match, the authentication will fail, and the MS will be rejected by the NS. We will focus on the 128-bit SSD because it must be either generated or updated first during the authentication and is related to *A-key*.

Before presenting the detailed description of the algorithm, we define the following symbols.

• If a variable A has n bits, where n is a multiple of 8, we can split it into bytes and denote those bytes by $A[i]$, $i = 0, ..., (\frac{n}{8}-1)$, from the most significant byte to the least significant byte. Note that the square bracket ([ ]) following a variable indicates that the variable is not the reference.

• The j-th bit of a variable A, where the value of the index j increases from the least significant bit (LSB) to the most significant bit (MSB), is denoted by A^j, $j \geqslant 0$. For example, if A is a byte, then A^7 is the MSB of A. $A^{j:k}$ represents $(j-k+1)$ consecutive bits of A starting from A^j, where $j > k \geqslant 0$ and $j, k \in \mathbb{Z}$.

• $\{\cdot||\cdot\}$ stands for the bitwise concatenation of two variables. For example, $\{A^{2:1}||B^{5:0}\}$ means that two bits of A and six bits of B are combined to form one byte.

• $\oplus$, $\vee$, $\wedge$ and $\neg$ are the bitwise XOR, OR, AND, and NOT operators, respectively.

• $>>$ is the logical right shift operator.

• $SX(A)$ is the self-XOR function, which applies the XOR operation to each bit of A. For example, if A is an n-bit variable, then $SX(A) = A^{n-1} \oplus A^{n-2}, ..., \oplus A^0$.

2.2 CAVE and SSD Generation

2.2.1 CAVE

CAVE is a hash-like function. Its primary components are a 32-bit linear feedback shift register (LFSR), 16 8-bit mixing registers (R_0–R_{15}), and a 256-entry nonlinear table (*CaveTable*). The contents of the LFSR are updated in each *LFSR_Update* procedure. In the procedure, the contents of the LFSR are first shifted one bit to the right, and the MSB of the LFSR is then calculated and associated with the other 4 bits of the LFSR, as shown in Fig.2.

```
LFSR_Update()
Procedure:
  $LFSR = LFSR >> 1$
  $LFSR[0]^7 = LFSR[1]^6 \oplus SX(LFSR[3]^{2:0})$
```

Fig.2. *LFSR_Update* procedure.

Before the execution of *CAVE*, the LFSR and the 16 mixing registers must be initialized. Then, *CAVE* will execute four or eight rounds of confusion operations. As Fig.3 shows, in each round, there are 16 internal iterations. Each internal iteration consists of two "while" loops, which are the most significant parts of *CAVE* for our SCA strategies. In each "while" loop, there is a lookup table operation. The lowest four bits or highest four bits (*lowNibble* or *highNibble*, respectively) of the CaveTable's output determine the condition of the following "if" statement. There are 16 possible values for *lowNibble* or *highNibble*. If the condition is true, then the LFSR will be updated once, and the "while" loop will run again. In contrast, the "while" loop will break if the condition is false. After the two "while" loops, *CAVE* will update the LFSR once and proceed to the next internal iteration. After the 16 internal iterations, the contents of R_0–R_{15} are rotated and shuffled. Finally, the 128-bit output is stored in R_0–R_{15}.

2.2.2 SSD Generation

The SSD generation procedure is presented in Fig.4. The inputs are a 56-bit *RANDSSD*, an 8-bit authentication algorithm version (*AAV*), a 32-bit *ESN* and a 64-bit *A-key*. Only *A-key* is secret.

The initialization of the LFSR and R_i is related to *A-key*. The 32 LSBs of *RANDSSD* and the 32 LSBs and 32 MSBs of *A-key* are XORed into the LFSR. *A-key* is loaded into R_0–R_7, and R_8 is initialized with AAV, which is fixed to 0xC7. The 24 MSBs of $RANDSSD$ are loaded into R_9, R_{10}, and R_{11}. *ESN* is loaded into R_{12}–R_{15}. The values of $offset1$ and $offset2$ are both set to 128. After initialization, an 8-round CAVE procedure runs. The output of *CAVE* is divided into the SSDA and the SSDB.

```
CAVE(number_of_rounds, offset1, offset2)
Input:
  number_of_rounds (4 or 8), offset1, offset2
Output:
  128-bit result in $R_0$ to $R_{15}$
Procedure:
  For round_index = number_of_rounds − 1 $\rightarrow$ 0:
    For $i$ = 0 $\rightarrow$ 15:
      While True:
        offset1 += (LFSR[0] $\oplus$ $R_i$) $\wedge$ 0xFF
        //lookup table, target of CPA
        lowNibble = CaveTable[offset1] $\wedge$ 0x0F
        //lowest 4 bits of $R_i$, target of SPA
        If lowNibble == $R_i$ $\wedge$ 0x0F:
          LFSR_Update();...
        Else: Break
      While True:
        offset2 += (LFSR[1] $\oplus$ $R_i$) $\wedge$ 0xFF
        //lookup table, target of CPA
        highNibble = CaveTable[offset2] $\wedge$ 0xF0
        //highest 4 bits of $R_i$, target of SPA
        If highNibble == $R_i$ $\wedge$ 0xF0:
          LFSR_Update();...
        Else: Break
      LFSR_Update();
    Postprocessing of $R_0$ to $R_{15}$
```

Fig.3. Simplified *CAVE* procedure.

```
SSD_Generation(RANDSSD, AAV, ESN, A-key)
Input:
  RANDSSD, AAV, ESN, A-key
Output:
  SSDA, SSDB
Procedure:
  LFSR[i] = RANDSSD[$i+3$]$\oplus$A-key[i]$\oplus$A-key[$i+4$],
  i=0,...,3
  If (LFSR[0]$\vee$LFSR[1]$\vee$LFSR[2]$\vee$LFSR[3]) == 0:
    LFSR[i] = RANDSSD[$i+3$], $i = 0, ..., 3$
  //A-key is stored in $R_0$ to $R_7$
  $R_i$ = A-key[i], $i = 0, ..., 7$
  $R_8$ = AAV;
  $R_i$ = RANDSSD[$i-9$], $i = 9, 10, 11$
  $R_i$ = ESN[$i-9$], $i = 12, ..., 15$;
  offset1 = offset2 = 128
  CAVE(8, offset1, offset2)
  SSDA[i] = $R_i$, $i = 0, ..., 7$;
  SSDB[i] = $R_{i+8}$, $i = 0, ..., 7$
```

Fig.4. SSD generation procedure.

2.3 AKA-Based Authentication Protocol

The AKA framework is illustrated in Fig.5. The 128-bit *A-key*, which acts as a root key, is the only secret data. $f1$, $f2$, $f3$, $f4$, $f5$, and $f11$ are six core functions used to generate a series of other keys and helper variables. They use different inputs and constants to initialize the SHA-1 algorithm (Subsection 2.4.1) and run SHA-1 to obtain a hash value, which will be post-processed to produce the corresponding outputs. The specific procedure of the $f5$ function is depicted in Subsection 2.4.2, and the other five functions are similar.

When an MS wants to join the network, it will send a request for registration to the NS. On the NS side, the NS will look up *A-key* corresponding to the identity of the MS and generate a 48-bit sequence number (*SQN*) and a 128-bit random number (*RAND*). *SQN* is a unique number of increasing values and is used to prevent replay attacks. *RAND* is a random challenge that is different for each registration procedure. A 16-bit authentication and key management field (*AMF*) is defined and used by the operators as helper data for several purposes (e.g., indicating the version of the algorithm used in the project). $f1$ uses *A-key*, *SQN*, *AMF*, and *RAND* as its inputs to generate a 64-bit message authentication code (*MAC*). $f5$ uses *RAND* and *A-key* as its inputs to generate a 48-bit anonymity key (*AK*). Then, $SQN \oplus AK$, *AMF*, and *MAC* are grouped to form an authentication token (*AUTN*), and the NS concatenates *AUTN* and *RAND* into AV that is sent to the MS. On the MS side, the MS extracts $SQN \oplus AK$, *AMF*, *MAC* and *RAND* from the received AV. $f5$ uses the received *RAND* and *A-key* to produce *AK*; then, $SQN \oplus AK$ is XORed with *AK* to obtain *SQN*. The MS will verify the freshness of *SQN* and the synchronization will fail if *SQN* is outdated. If *SQN* is fresh, then the MS will run $f1$ with *SQN*, the received *RAND*, the received *AMF*, and *A-key* to generate a 64-bit expected message authentication code (*XMAC*). If its *XMAC* is not equal to the received *MAC*, it means that the inputs to the $f1$ function were different on each side, which may have been caused by either data corruption during the transmission phase (resulting in a different AV) or an illegal NS (resulting in different *A-key*). In this case, the MS will reject the NS and terminate the authentication procedure. If *XMAC* is equal to *MAC*, then the MS will proceed to execute the $f2$ function with the received *RAND* and *A-key*. The output of $f2$ is the response (*RES*) to be sent to the NS. The NS will compare the received *RES* to its expected user response (*XRES*) generated by $f2$ with *RAND* and *A-key* on the NS side. If *XRES* is not equal to *RES*, the NS will reject the MS and terminate the authentication procedure since the inputs to $f2$ on each side were different. If *XRES*

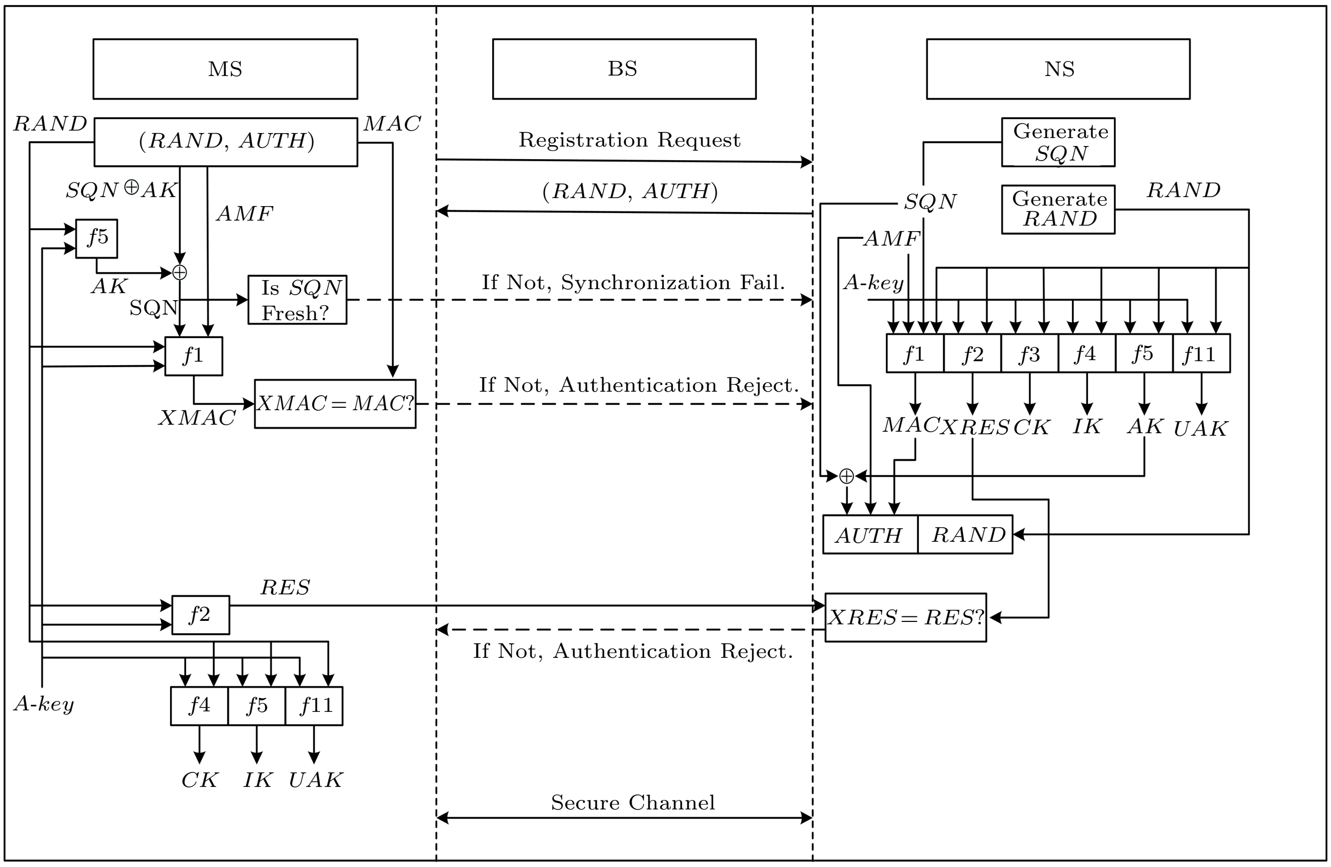


Fig.5. AKA-based authentication protocol.

is equal to *RES*, the mutual authentication will succeed, and a secure channel will be established. Then, the MS and the NS will run $f3$, $f4$, and $f11$ with the same *RAND* and the same *A-key* to generate a ciphering key (*CK*), an integrity key (*IK*) and a user identity module (UIM) authentication key (*UAK*) respectively to protect the communication data through the secure channel.

From the above description, we can see that $f1$, $f2$ and $f5$ are related to the authentication process, whereas $f3$, $f4$ and $f11$ are related to the secure communication. For each core function, in addition to the inputs mentioned above, there are two more constants used during execution. One is a 32-bit family key (*Fmk*), which is the same for each core function (0x41484147). The other is an 8-bit type identifier (TI), which is unique to each core function. The *TI*s of $f1$, $f2$, $f3$, $f4$, $f5$ and $f11$ are 0x42, 0x44, 0x45, 0x46, 0x47 and 0x50, respectively. Since $f5$ is the first function executed by the SIM card (on MS side) during the authentication process and its inputs are related to *A-key*, we select $f5$ as our CPA's target to recover *A-key*.

2.4 SHA-1 and the $f5$ Procedure

2.4.1 SHA-1

SHA-1 is a hash function that operates on a 512-bit payload (message block) and produces a 160-bit message digest. In CDMA2000, only one message block is processed at a time. SHA-1 has several primary variables, constants, and operations, as follows.

• The variables $W_t, 0 \leqslant t \leqslant 79$, represent a message schedule for 80 32-bit words.

• a, b, c, d, and e denote five 32-bit working variables.

• $H^{(i)} = \{H_0^{(i)}, H_1^{(i)}, H_2^{(i)}, H_3^{(i)}, H_4^{(i)}\}$, $i = 0, ..., N$ represents five 32-bit words in the form of hash values, where N is the number of message blocks ($N = 1$ in CDMA2000) and $H^{(0)}$ is the initial hash value (IV).

• $M = \{M_0, ..., M_{15}\}$ denotes a payload (message block) consisting of 16 32-bit words.

• The constants K_t, $0 \leqslant t \leqslant 79$, represent 80 constant 32-bit words.

• $ROTL^n(x)$ denotes the circular left shift operation, where x is a w-bit variable and n is an integer such that $0 \leqslant n < w$.

• The functions $f_t(x, y, z)$, $0 \leqslant t \leqslant 79$, are a series of logical functions, each with an input consisting of three 32-bit words and an output consisting of a 32-bit word. As mentioned in the SHA-1 standard[10], the XOR operation ($\oplus$) in these functions may be replaced with a bitwise OR operation ($\vee$), in which case the functions will produce identical results. Hence, the sample code for SHA-1 in the specification uses $\vee$ instead of $\oplus$.

$$f_t(x,y,z) = \begin{cases} Ch(x,y,z) = (x \wedge y) \oplus (\neg x \wedge z), & \text{if } 0 \leqslant t \leqslant 19, \\ Parity(x,y,z) = x \oplus y \oplus z, & \text{if } 20 \leqslant t \leqslant 39, \\ Maj(x,y,j) = (x \wedge y) \oplus (x \wedge z) & \\ \oplus (y \wedge z), & \text{if } 40 \leqslant t \leqslant 59, \\ Parity(x,y,z) = x \oplus y \oplus z, & \text{if } 60 \leqslant t \leqslant 79. \end{cases}$$

The IV and the payload must be initialized before the execution of SHA-1. After the initialization, there will be 80 rounds of operation. Finally, a hash value ($H^{(1)}$) is outputted as the message digest. The detailed procedure is shown in Fig.6[10].

```
SHA-1($H^{(0)}$, $M_i$)
Input:
  $H^{(0)}$, $M_i (i = 0, ..., 15)$
Output:
  $H^{(1)}$
Procedure:
  $W_t = \begin{cases} M_t, & \text{if } 0 \leqslant t \leqslant 15, \\ ROTL^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}), & \text{if } 16 \leqslant t \leqslant 79, \end{cases}$
  $a = H_0^{(0)}; b = H_1^{(0)}; c = H_2^{(0)}; e = H_3^{(0)}; f = H_4^{(0)}$
  For $t = 0 \to 79$:
    $T = ROTL^5(a) + f_t(b,c,d) + e + K_t + W_t$;
    $e = d$; $d = c$; $c = ROTL^{30}(b)$; $b = a$; $a = T$
  $H_0^{(1)} = a + H_0^{(0)}$; $H_1^{(1)} = b + H_1^{(0)}$; $H_2^{(1)} = c + H_2^{(0)}$;
  $H_3^{(1)} = d + H_3^{(0)}$; $H_4^{(1)} = e + H_4^{(0)}$;
```

Fig.6. One-block SHA-1 procedure.

2.4.2 *f*5 *Procedure*

The $f5$ procedure is presented in Fig.7. The 128-bit *A-key* and the 128-bit *RAND* are the inputs of $f5$. The constant *Fmk* (0x41484147) and the constant *TI* of $f5$ (0x47) are also used in this procedure. First, the 160-bit IV and the 512-bit payload are initialized. The IV is loaded with the standard initial hash value[10] and the payload is loaded with the constant 0x5C repeated 64 times. Then, *A-key* is XORed into the leftmost 128 bits of the IV. *TI* is XORed into the 11th byte of the payload. *Fmk* is XORed into the 12th to the 15th bytes of the payload. *RAND* is XORed into the 16th to the 31st bytes of the payload. Then, $f5$ executes SHA-1 to produce a 160-bit hash value. *AK* is obtained by postprocessing this hash value.

```
f5(A-key, RAND)
Input:
  A-key (A-key$_i$, $i = 0,1,2,3$), RAND (RAND$_i$, $i = 0,1,2,3$),
  (each A-key$_i$ and RAND$_i$ are 32-bit words)
Output:
  AK
Procedure:
  Fmk= 0x41484147
  TI= 0x47
  $H_0^{(0)}$ = 0x67452301, $H_1^{(0)}$ = 0xEFCDAB89, $H_2^{(0)}$ = 0x98BADCFE,
  $H_3^{(0)}$ = 0x10325476, $H_4^{(0)}$ = 0xC3D2E1F0
  $M_i$ = 0x5C5C5C5C, $i = 0, ..., 15$
  $H_i^{(0)} = H_i^{(0)} \oplus$ A-key$_i$, $i = 0, ..., 3$
  $M_2[3] = M_2[3] \oplus$ TI
  $M_3 = M_3 \oplus$ Fmk
  $M_i = M_i \oplus$ RAND$_{i-4}$, $i = 4, ..., 7$
  $H^{(1)}$ = SHA-1($H^{(0)}$, $M_i$)
  AK = postprocessing($H^{(1)}$)
```

Fig.7. $f5$ procedure.

3 SCA for SSD Generation and $f5$ Procedure

3.1 SCA

SCA[5] exploits the vulnerability of an implementation of a cryptographic algorithm by utilizing various pieces of side-channel information, such as power consumption, electromagnetic (EM) radiation, and time of execution, which are produced by a device on which the algorithm is executed[5]. The side-channel information at a particular moment depends on the operands and the operations being performed at that time. Based on the physical leakage of those data-dependent and operation-dependent values, SCA can be applied either to recover the key directly or to recover intermediate key-related variables that are assumed to be invisible from a mathematical point of view. With such intermediate key-related data, it is easier to extract the correct key that is expected based on traditional cryptographic analysis, in which it is assumed that the algorithm is a black box. Although we consider power consumption measurements in this paper, the strategies we propose are general and are not limited to power consumption.

3.1.1 *SPA*

SPA[11] offers a brief and effective means of inferring secret data by directly observing the power consumption of a device. Since there is a relationship between

[10]FIPS, PUB. 180-4 Secure hash standard (SHS), Mar. 2012.

the operation being performed and the power consumption, the position of the target operation can be approximately located based on the pattern of the algorithm. Moreover, if the algorithm has unbalanced branches, meaning that the execution time of different branches is different, then this imbalance may be observed in the measurements. Consequently, the resulting distinct pattern can be utilized to infer the condition of a branch from a single power trace. We use SPA to locate the position and recover part of *A-key* of the CAVE-based protocol.

3.1.2 CPA

CPA[10] is a variant of differential power analysis (DPA)[11]. It can exploit the dependency between the actual physical leakages and the hypothetical leakages of intermediate data related to the key on the basis of a statistical measure called the Pearson correlation coefficient. Commonly, a physical leakage is related to only part of the key due to byte-oriented or word-oriented operations on the device. For this reason, a divide-and-conquer strategy can be used for key recovery in CPA. We use CPA to recover key-related variables in both the CAVE-based and the AKA-based protocols.

3.2 SCA for SSD Generation

In this subsection, we propose a scheme for recovering *A-key* used in the SSD generation procedure. For a SIM card, *RANDSSD* is the external input that we can control. Given a known *RANDSSD*, the first eight iterations of the first outer loop of the CAVE algorithm need to be analyzed to recover *A-key*. We will first describe the detailed analysis procedure for the first iteration and then generalize the analysis to the remaining seven iterations.

In the first "while" loop of the first iteration, a lookup table operation is performed. Since the CaveTable is a bijective map, the hamming weight of its output could be the target of CPA. As shown in Subsection 2.2.2, the entry of the CaveTable is LFSR[0] $\oplus R_0 + offset1$. After the initialization, $offset1 = 128$, LFSR[0] $= RANDSSD[3] \oplus A\text{-}key[0] \oplus A\text{-}key[4]$, and $R_0 = A\text{-}key[0]$, thereby the entry is actually $RANDSSD[3] \oplus A\text{-}key[4] + 128$. *RANDSSD*[3] can be treated as a random and known variable since it can be controlled from the outside. However, *A-key*[0] is always a constant and unknown variable. We can regard the fixed part (*A-key*[0]) as a "key" and the random part as a "plain text" which is known. The "key" and the "plain text" are XORed, and the result is related to the entry of the bijective map. This is the classic attack scenario of CPA. We can perform an exhaustive search of the possible "key" values and calculate the corresponding outputs from the CaveTable with the known *RANDSSD*[3] and the known $offset1$. Then, we can obtain the hamming weight of each possible output. Finally, via the CPA, the correct "key" value (*A-key*[4]) can be obtained.

With the known "key" value, we can calculate the correct *lowNibble* value for each power trace. As described in Subsection 2.2.1, *lowNibble* has 16 possible values, and one of them is equal to $R_0 \wedge$ 0x0F. The total trace set can be divided into 16 groups according to the *lowNibble* value of each trace. Only one group's *lowNibble* value, i.e., the one that is equal to the lowest four bits of R_0, makes the condition true. On the true branch, the LFSR is updated once, and the "while" loop runs again. The other 15 groups belong to the false branch, which exits the "while" loop. These two cases correspond to different patterns in the power traces. We can calculate the mean trace for each group, and one of these 16 mean traces will be different from the other 15 mean traces. The *lowNibble* value of the distinct group will be the exactly correct value of the lowest four bits of R_0. By combining the results of CPA and SPA, we can obtain *A-key*[4] and the lowest four bits of *A-key*[0]. At this time, the LFSR will have been updated at least once in the distinct trace group, and no update will have yet been performed for the other 15 groups. The distinct group must be filtered out since the corresponding traces would complicate the subsequent analysis. Thus, the new trace set is composed of the remaining 15 groups.

For the second "while" loop of the first iteration, LFSR[1] $\oplus R_0 = RANDSSD[4] \oplus A\text{-}key[1] \oplus A\text{-}key[5] \oplus A\text{-}key[0]$. This can again be divided into a "key" ($A\text{-}key[1] \oplus A\text{-}key[5] \oplus A\text{-}key[0]$) and "plain text" (*RANDSSD*[4]), and we can obtain the "key" value via CPA based on the remaining traces. Similar to the previous analysis, we can perform SPA to identify the highest four bits of R_0. Consequently, *A-key*[0] will be known (the lowest and highest four bits of R_0). Then, we can determine $A\text{-}key[1] \oplus A\text{-}key[5]$. Finally, to ensure that the LFSR is not updated during the "while" loop, we again filter out the traces corresponding to the true branch. After these two "while" loops, the LFSR will be updated once, and the algorithm will proceed to the next iteration.

The analysis procedure for the rest seven iterations is similar to those described above. For the convenience of description, we define the following notations.

• IR_i, $i = 0, ..., 7$, denotes $(i+1)$-th iteration of the first outer loop.

• $IR_{i.0}$ and $IR_{i.1}$ denote the first and the second "while" loops of IR_i respectively.

• $LSFR_i$ denotes the status of the LSFR at the beginning of IR_i.

• $IR_{i.j.p}$, $j = 0, 1$, denotes the random part of $LFSR_i[j] \oplus R_i$ in $IR_{i.j}$, which acts as the "plain text" for $IR_{i.j}$.

• $IR_{i.j.k}$, $j = 0, 1$, denotes the fixed part of $LFSR_i[j] \oplus R_i$ in $IR_{i.j}$, which acts as the "key" for $IR_{i.j}$, i.e., the CPA result.

• $IR_{i.j.n}$ denotes the exact nibble that makes the "if" statement true, i.e., the SPA result.

• *offset*1_i and *offset*2_i denote the values of *offset*1 and *offset*2, at the beginning of IR_i respectively.

• UB^i denotes the fixed part of the MSB of $LFSR_i$.

When analyzing $IR_{i.j}$ ($i = 0, ..., 7$, $j = 0, 1$), the first task is calculating the status of $LFSR_i[j]$ and R_i. $LFSR_i[j]$ can be derived according to its update procedure, and R_i is always equal to *A-key*[*i*]. UB^i is a mnemonic bit that stores the fixed part of the MSB of $LFSR_i$. Then, $LFSR_i[j] \oplus R_i$ can be divided into $IR_{i.j.k}$ and $IR_{i.j.p}$. $IR_{i.j.k}$ is the secret data whose correct value can be obtained through the CPA. We can calculate the value of $offset1_{i+1}$ (or $offset2_{i+1}$) with $IR_{i.0.k}$ (or $IR_{i.1.k}$), $IR_{i.0.p}$ (or $IR_{i.1.p}$) and $offset1_i$ (or $offset2_i$). Then, the SPA can be performed to obtain the value of $IR_{i.j.n}$. $IR_{i.0.n}$ (or $IR_{i.1.n}$) is always equal to the lowest (or highest) four bits of *A-key*[*i*]. Table 1 and Table 2 list the statuses of $LFSR_i[j]$, UB^i, $IR_{i.j.p}$ and $IR_{i.j.k}$. In these two tables, *RANDSSD* and *A-key* are represented by *RD* and *K* respectively.

In the *i*-th iteration, we can derive some parts of *A-key* and UB^i with the known $IR_{i.j.k}$ and the known $IR_{i.j.n}$. Since UB^i is the fixed part of $LFSR_i[0]^7$ and $IR_{i.0.k}$ is the fixed part of $LFSR_i[0] \oplus R_i$, we can know that $UB^i = IR^7_{i.0.k} \oplus R[i]^7 = IR^7_{i.0.k} \oplus A\text{-}key[i]^7$. In

Table 1. Statuses of $LFSR_i[0]$, UB^i, $IR_{i,0,p}$, and $IR_{i,0,k}$

i	$LFSR_i[0]$	UB^i	$IR_{i,0,p}$	$IR_{i,0,k}$
0	$RD[3] \oplus K[0] \oplus K[4]$	$(RD[3] \oplus K[0] \oplus K[4])^7$	$RD[3]$	$K[4]$
1	$\{(RD[4]^6 \oplus UB^1) \Vert (RD[3] \oplus K[0] \oplus K[4])^{7:1}\}$	$K[1]^6 \oplus K[5]^6 \oplus \mathrm{SX}((RD[6] \oplus K[3] \oplus K[7])^{2:0})$	$\{RD[4]^6 \Vert RD[3]^{7:1}\}$	$K[1] \oplus \{UB^1 \Vert (K[0] \oplus K[4])^{7:1}\}$
2	$\{(RD[4]^{7:6} \oplus UB^{2:1}) \Vert (RD[3] \oplus K[0] \oplus K[4])^{7:2}\}$	$K[1]^7 \oplus K[5]^7 \oplus \mathrm{SX}((RD[6] \oplus K[3] \oplus K[7])^{3:1})$	$\{RD[4]^{7:6} \Vert RD[3]^{7:2}\}$	$K[2] \oplus \{UB^{2:1} \Vert (K[0] \oplus K[4])^{7:2}\}$
3	$\{(\{RD[3]^0 \Vert RD[4]^{7:6}\} \oplus UB^{3:1}) \Vert (RD[3] \oplus K[0] \oplus K[4])^{7:3}\}$	$K[0]^0 \oplus K[4]^0 \oplus \mathrm{SX}((RD[6] \oplus K[3] \oplus K[7])^{4:2})$	$\{RD[3]^0 \Vert RD[4]^{7:6} \Vert RD[3]^{7:3}\}$	$K[3] \oplus \{UB^{3:1} \Vert (K[0] \oplus K[4])^{7:3}\}$
4	$\{(RD[3]^{1:0} \oplus UB^{4:3}) \Vert (RD[4]^{7:6} \oplus UB^{2:1}) \Vert (RD[3] \oplus K[0] \oplus K[4])^{7:4}\}$	$K[0]^1 \oplus K[4]^1 \oplus \mathrm{SX}((RD[6] \oplus K[3] \oplus K[7])^{5:3})$	$\{RD[3]^{1:0} \Vert RD[4]^{7:6} \Vert RD[3]^{7:4}\}$	$K[4] \oplus \{UB^{4:1} \Vert (K[0] \oplus K[4])^{7:4}\}$
5	$\{(RD[3]^{2:0} \oplus UB^{5:3}) \Vert (RD[4]^{7:6} \oplus UB^{2:1}) \Vert (RD[3] \oplus K[0] \oplus K[4])^{7:5}\}$	$K[0]^2 \oplus K[4]^2 \oplus \mathrm{SX}((RD[6] \oplus K[3] \oplus K[7])^{6:4})$	$\{RD[3]^{2:0} \Vert RD[4]^{7:6} \Vert RD[3]^{7:5}\}$	$K[5] \oplus \{UB^{5:1} \Vert (K[0] \oplus K[4])^{7:5}\}$
6	$\{(RD[3]^{3:0} \oplus UB^{6:3}) \Vert (RD[4]^{7:6} \oplus UB^{2:1}) \Vert (RD[3] \oplus K[0] \oplus K[4])^{7:6}\}$	$K[0]^3 \oplus K[4]^3 \oplus \mathrm{SX}((RD[6] \oplus K[3] \oplus K[7])^{7:5})$	$\{RD[3]^{3:0} \Vert RD[4]^{7:6} \Vert RD[3]^{7:6}\}$	$K \oplus [6]\{UB^{6:1} \Vert (K[0] \oplus K[4])^{7:6}\}$
7	$\{(RD[3]^{4:0} \oplus UB^{7:3}) \Vert (RD[4]^{7:6} \oplus UB^{2:1}) \Vert (RD[3] \oplus K[0] \oplus K[4])^7\}$	$K[0]^4 \oplus K[4]^4 \oplus RD[5]^0 \oplus K[2]^0 \oplus K[6]^0 \oplus \mathrm{SX}((RD[6] \oplus K[3] \oplus K[7])^{7:6})$	$\{RD[3]^{4:0} \Vert RD[4]^{7:6} \Vert RD[3]^7\}$	$K[7] \oplus \{UB^{7:1} \Vert (K[0] \oplus K[4])^7\}$

Table 2. Statuses of $LFSR_i[1]$, $IR_{i,1,p}$, and $IR_{i,1,k}$

i	$LFSR_i[1]$	$IR_{i,1,p}$	$IR_{i,1,k}$
0	$RD[4] \oplus K[1] \oplus K[5]$	$RD[4]$	$K[1] \oplus K[5] \oplus K[0]$
1	$\{(RD[3] \oplus K[0] \oplus K[4])^0 \Vert (RD[4] \oplus K[1] \oplus K[5])^{7:1}\}$	$\{RD[3]^0 \Vert RD[4]^{7:1}\}$	$\{(K[0] \oplus K[4])^0 \Vert (K[1] \oplus K[5])^{7:1}\} \oplus K[1]$
2	$\{(RD[3] \oplus K[0] \oplus K[4])^{1:0} \Vert (RD[4] \oplus K[1] \oplus K[5])^{7:2}\}$	$\{RD[3]^{1:0} \Vert RD[4]^{7:2}\}$	$\{(K[0] \oplus K[4])^{1:0} \Vert (K[1] \oplus K[5])^{7:2}\} \oplus K[2]$
3	$\{(RD[3] \oplus K[0] \oplus K[4])^{2:0} \Vert (RD[4] \oplus K[1] \oplus K[5])^{7:3}\}$	$\{RD[3]^{2:0} \Vert RD[4]^{7:3}\}$	$\{(K[0] \oplus K[4])^{2:0} \Vert (K[1] \oplus K[5])^{7:3}\} \oplus K[3]$
4	$\{(RD[3] \oplus K[0] \oplus K[4])^{3:0} \Vert (RD[4] \oplus K[1] \oplus K[5])^{7:4}\}$	$\{RD[3]^{3:0} \Vert RD[4]^{7:4}\}$	$\{(K[0] \oplus K[4])^{3:0} \Vert (K[1] \oplus K[5])^{7:4}\} \oplus K[4]$
5	$\{(RD[3] \oplus K[0] \oplus K[4])^{4:0} \Vert (RD[4] \oplus K[1] \oplus K[5])^{7:5}\}$	$\{RD[3]^{4:0} \Vert RD[4]^{7:5}\}$	$\{(K[0] \oplus K[4])^{4:0} \Vert (K[1] \oplus K[5])^{7:5}\} \oplus K[5]$
6	$\{(RD[3] \oplus K[0] \oplus K[4])^{5:0} \Vert (RD[4] \oplus K[1] \oplus K[5])^{7:6}\}$	$\{RD[3]^{5:0} \Vert RD[4]^{7:6}\}$	$\{(K[0] \oplus K[4])^{5:0} \Vert (K[1] \oplus K[5])^{7:6}\} \oplus K[6]$
7	$\{(RD[3] \oplus K[0] \oplus K[4])^{6:0} \Vert (RD[4] \oplus K[1] \oplus K[5])^7\}$	$\{RD[3]^{6:0} \Vert RD[4]^7\}$	$\{(K[0] \oplus K[4])^{6:0} \Vert (K[1] \oplus K[5])^7\} \oplus K[7]$

IR_0, we can get:

$$\begin{cases} A\text{-}key[4]=IR_{0.0.k}, \\ A\text{-}key[0]=IR_{0.0.n}\vee IR_{0.1.n}, \\ A\text{-}key[1]\oplus A\text{-}key[5]=IR_{0.1.k} \oplus A\text{-}key[0]. \end{cases}$$

A-$key[1]\oplus A$-$key[5]$ can be used in IR_1, and we can obtain A-$key[1]$ and A-$key[5]$,

$$\begin{cases} A\text{-}key[1]=IR_{1.0.n}\vee IR_{1.1.n} \\ \qquad =\{(IR_{1.1.n}{}^{7})||(IR_{1.0.k}{}^{6:0}\oplus \\ \qquad\quad (A\text{-}key[0]\oplus A\text{-}key[4])^{7:1})\}, \\ A\text{-}key[5]=(A\text{-}key[1]\oplus A\text{-}key[5])\oplus A\text{-}key[1]. \end{cases}$$

In $IR_i, i=2,...,7$, we can get A-$key[i]$:

$$\begin{cases} A\text{-}key[2]=IR_{2.0.n}\vee IR_{2.1.n} \\ \qquad =\{(IR_{2.1.n}{}^{7})||(IR_{2.0.k}{}^{6:0}\oplus \\ \qquad\quad \{UB^{1}||(A\text{-}key[0]\oplus A\text{-}key[4])^{7:2}\})\}, \\ A\text{-}key[3]=IR_{2.0.n}\vee IR_{2.1.n} \\ \qquad =\{(IR_{3.1.n}{}^{7})||(IR_{3.0.k}{}^{6:0}\oplus \\ \qquad\quad \{UB^{2:1}||(A\text{-}key[0]\oplus A\text{-}key[4])^{7:3}\})\}, \\ A\text{-}key[4]=IR_{4.0.n}\vee IR_{4.1.n} \\ \qquad =\{(IR_{4.1.n}{}^{7})||(IR_{4.0.k}{}^{6:0}\oplus \\ \qquad\quad \{UB^{3:1}||(A\text{-}key[0]\oplus A\text{-}key[4])^{7:4}\})\}, \\ A\text{-}key[5]=IR_{5.0.n}\vee IR_{5.1.n} \\ \qquad =\{(IR_{5.1.n}{}^{7})\ ||(IR_{5.0.k}{}^{6:0}\oplus \\ \qquad\quad \{UB^{4:1}||(A\text{-}key[0]\oplus A\text{-}key[4])^{7:5}\})\}, \\ A\text{-}key[6]=\ IR_{6.0.n}\vee IR_{6.1.n} \\ \qquad =\{(IR_{6.1.n}{}^{7})||(IR_{6.0.k}{}^{6:0}\oplus \\ \qquad\quad \{UB^{5:1}||(A\text{-}key[0]\oplus A\text{-}key[4])^{7:6}\})\}, \\ A\text{-}key[7]=\ IR_{7.0.n}\vee IR_{7.1.n} \\ \qquad =\{(IR_{7.1.n}{}^{7})||(IR_{7.0.k}{}^{6:0}\oplus \\ \qquad\quad \{UB^{6:1}||(A\text{-}key[0]\oplus A\text{-}key[4])^{7}\})\}. \end{cases}.$$

Since A-$key[4]$ and A-$key[5]$ are already known in IR_0 and IR_1 respectively, we do not need to perform CPA and SPA in IR_4 and IR_5 respectively. However, we still need to filter out the traces in these two iterations. In IR_i, $i = 1, 2, 3, 6, 7$, there are two ways to recover A-$key[i]$; one is to combine $IR_{i.0.k}$ and $IR_{i.1.n}$, and the other is to combine $IR_{i.0.n}$ and $IR_{i.1.n}$. This redundancy provides us with an opportunity for a double check. According to Table 1 and Table 2, only *RANDSSD*[3] and *RANDSSD*[4] need to be random. The other bytes of *RANDSSD* can be fixed to simplify the analysis. To obtain the full A-key, 12 rounds of CPA and 12 rounds of SPA are needed. Suppose that the total number of traces in the original trace set is N and that one round of CPA requires m traces. There are 15 rounds of filtration and each filtration will reduce the current trace set by $\frac{1}{16}$. To ensure that sufficient traces will remain for the last round of CPA, N and m must satisfy the following requirement:

$$N \times (\frac{15}{16})^{15} \geqslant m.$$

3.3 SCA for the $f5$ Procedure

In this subsection, we propose a strategy for recovering A-key used in the $f5$ function. Similar to the approach for the SSD generation procedure, we can know and control the 128-bit *RAND*. As Fig.7 shows, *RAND* is XORed with the 16th to the 31st bytes of the payload, which are represented as four words (M_4, M_5, M_6, and M_7). In SHA-1, M_0–M_{15} are assigned to W_0–W_{15} respectively. Since M_0–M_3 are fixed, all variables in the first four rounds are fixed, and they cannot be used for CPA. Instead, we must analyze starting from at least the 5th round ($t = 4$).

In each round, T will be updated first, and the new T will be assigned to a. Therefore, at the beginning of round t ($t \geqslant 1$), a is T from round $t-1$. We have the following expression:

$$T = ROTL^5(a) + f_t(b, c, d) + e + K_t + W_t.$$

When $t = 4$, T can be divided into a fixed part ($ROTL^5(a) + f_t(b, c, d) + e$) and a random part ($W_t$), where a is T from round 3. Theoretically, with the known random part and the secret fixed part, we can apply CPA to this addition operation. However, it is a 32-bit addition, and we experimentally find it difficult to successfully apply CPA to this operation. Thus, we need to find another target. Let us look at the ROTL operation. The ROTL operation is a circular left shift operation, and it seems that the hamming weight of ROTL's output does not change compared with the hamming weight of ROTL's input. However, for an MCU, ROTL is not a primitive operation. This means that the ROTL operation will be divided into several other operations for actual execution. We find the implementation of ROTL in the C language in the specification and compile it using avr-gcc. We choose the

ATmega128A, an 8-bit MCU, as our target chip. The C code of ROTL and the assembly code of $ROTL^5(a)$ with the default compilation settings are presented in Fig.8.

```
// Implementation of $ROTL^n(a)$ in C
#define S(a,n) ((a << n) | (a >> (32 - n)))
// Assembly of $ROTL^5(a)$
// $R_{24}$-$R_{27}$ store the $a$
LDS R24,0x0116   Load direct from data space;
LDS R25,0x0117   Load direct from data space;
LDS R26,0x0118   Load direct from data space;
LDS R27,0x0119   Load direct from data space;
//$R_{24}$-$R_{27}$ are copied to $R_{20}$-$R_{23}$
MOVW R20,R24    Copy register pair ;
MOVW R22,R26  Copy register pair ;
// 5 times 1-bit left shift
LSL R20   Logical shift left;
ROL R21   Rotate left through carry;
ROL R22   Rotate left through carry;
ROL R23   Rotate left through carry;
LSL R20   Logical shift left;
ROL R21   Rotate left through carry;
ROL R22   Rotate left through carry;
ROL R23   Rotate left through carry;
LSL R20   Logical shift left;
ROL R21   Rotate left through carry;
ROL R22   Rotate left through carry;
ROL R23   Rotate left through carry;
LSL R20   Logical shift left;
ROL R21   Rotate left through carry;
ROL R22   Rotate left through carry;
ROL R23   Rotate left through carry;
LSL R20   Logical shift left;
ROL R21   Rotate left through carry;
ROL R22   Rotate left through carry;
ROL R23   Rotate left through carry;
MOV R0,R23    Copy register;
// 27 times 1-bit right shift
LDI R23,0x1B    Load immediate;
LSR R27   Logical shift right;
ROR R26   Rotate right through carry;
ROR R25   Rotate right through carry;
ROR R24   Rotate right through carry;
DEC R23   Decrement;
BRNE PC-0x05    Branch if not equal;
MOV R23,R0    Copy register;
// Or the results of left shift and right shift
OR R24,R20    Logical OR;
OR R25,R21    Logical OR;
OR R26,R22    Logical OR;
OR R27,R23    Logical OR;
```

Fig.8. C code and assembly code of $ROTL^5(a)$.

We can see that for $ROTL^5(a)$, the 1-bit left shift will be performed five times, and the 1-bit right shift will be performed 27 times. The hamming weight of each register will change during the execution of $ROTL^5(a)$. Therefore, we can choose the result of $ROTL^5(a)$ as our CPA target and perform byte-wise CPA 4 times to recover the fixed part of a for an 8-bit device. If we want to know the fixed part of T from round t, we should apply the above method to $ROTL^5(a)$ from round $t+1$.

For a 16-bit or 32-bit device, the only difference is the length of the register; the ROTL operation will still be divided into left shift operations and right shift operations. This difference simply increases the space to be exhausted; the above strategy remains effective.

Next, we will describe how to recover *A-key* via CPA for $ROTL^5(a)$ in specific rounds. First, we define some notations as follows:

- RD_t, $t = 0, ..., 79$ denotes the t-th round of SHA-1.
- $A\text{-}key_i$, $i = 0, ..., 3$ denotes each 32-bit word of *A-key* from the most significant word to the least significant word.
- T_t, a_t, b_t, c_t, d_t, and e_t, $t = 0, ..., 79$, denote the final statuses of T, a, b, c, d, and e at the end of RD_t respectively.
- $T_{t.p}$ denotes the random part of T_t.
- $T_{t.k}$ denotes the fixed part of T_t.

As shown in Subsection 2.4, the initial values of a, b, c, d and e are as follows:

$$\begin{cases} a = H_0^{(0)} = 67452301 \oplus A\text{-}key_0, \\ b = H_1^{(0)} = \text{EFCDAB89} \oplus A\text{-}key_1, \\ c = H_2^{(0)} = \text{98BADCFE} \oplus A\text{-}key_2, \\ d = H_3^{(0)} = 10325476 \oplus A\text{-}key_3, \\ e = H_4^{(0)} = \text{C3D2E1F0}. \end{cases}$$

We can see that a, b, c, and d are related to *A-key*. Therefore, the final goal is to recover them so that we can infer *A-key*. Table 3 lists the statuses of a_t, b_t, c_t, d_t, e_t, and the unfolded T_t, $t = 0, ..., 11$.

In RD_4, W_4 is random and denoted by $T_{4.p}$, and $ROTL^5(T_3) + K_4 + e_3 + ((T_2 \wedge c_3) \vee (\neg T_2 \wedge d_3))$ is fixed and denoted by $T_{4.k}$; thus, $T_4 = T_{4.p} + T_{4.k}$. In RD_5, $ROTL^5(T_4)$ is executed; this is the target of CPA for the recovery of $T_{4.k}$. We must perform 8-bit CPA four times. The first round of CPA is performed to recover $T_{4.k}^{26:19}$. Since there is an addition operation, if $T_{4.p}^{18:0} + T_{4.k}^{18:0}$ has a carry, then the CPA result will be affected. To prevent this situation, we need to force $T_{4.p}^{18:0}$ and $T_{4.p}^{31:27}$ to be 0. Since W_4 is equal to M_4, which is initialized with 0x5C5C5C5C and is XORed with $RAND_0$, we can load $RAND_0$ with 0x5C5C5C5C and then make $RAND_0^{26:19}$ random. With this chosen $RAND$, we can efficiently recover $T_{4.k}^{26:19}$. We can also load $RAND_0$ with 0x5C5C5C5C and separately make $RAND_0^{18:11}$,

Table 3. Statuses of a_t, b_t, c_t, d_t, e_t, and T_t in the First 12 Rounds

t	a_t	b_t	c_t	d_t	e_t	T_t
0	T_0	a	$ROTL^{30}(b)$	c	d	$ROTL^5(a) + K_0 + e + W_0 + ((b \wedge c) \vee (\neg b \wedge d))$
1	T_1	T_0	$ROTL^{30}(a)$	$ROTL^{30}(b)$	c	$ROTL^5(T_0) + K_1 + d + W_1 + ((b_0 \wedge c_0) \vee (\neg b_0 \wedge d_0))$
2	T_2	T_1	$ROTL^{30}(T_0)$	$ROTL^{30}(a)$	$ROTL^{30}(b)$	$ROTL^5(T_1) + K_2 + c + W_2 + ((T_0 \wedge c_1) \vee (\neg T_0 \wedge d_1))$
3	T_3	T_2	$ROTL^{30}(T_1)$	$ROTL^{30}(T_0)$	$ROTL^{30}(a)$	$ROTL^5(T_2)+K_3+ROTL^{30}(b)+W_3+((T_1 \wedge c_2) \vee (\neg T_1 \wedge d_2))$
4	T_4	T_3	$ROTL^{30}(T_2)$	$ROTL^{30}(T_1)$	$ROTL^{30}(T_0)$	$ROTL^5(T_3)+K_4+ROTL^{30}(a)+W_4+((T_2 \wedge ROTL^{30}(T_1)) \vee (\neg T_2 \wedge ROTL^{30}(T_0)))$
5	T_5	T_4	$ROTL^{30}(T_3)$	$ROTL^{30}(T_2)$	$ROTL^{30}(T_1)$	$ROTL^5(T_4)+K_5+ROTL^{30}(T_0)+W_5+((T_3 \wedge ROTL^{30}(T_2)) \vee (\neg T_3 \wedge ROTL^{30}(T_1)))$
6	T_6	T_5	$ROTL^{30}(T_4)$	$ROTL^{30}(T_3)$	$ROTL^{30}(T_2)$	$ROTL^5(T_5)+K_6+ROTL^{30}(T_1)+W_6+((T_4 \wedge ROTL^{30}(T_3)) \vee (\neg T_4 \wedge ROTL^{30}(T_2)))$
7	T_7	T_6	$ROTL^{30}(T_5)$	$ROTL^{30}(T_4)$	$ROTL^{30}(T_3)$	$ROTL^5(T_6)+K_7+ROTL^{30}(T_2)+W_7+((T_5 \wedge ROTL^{30}(T_4)) \vee (\neg T_5 \wedge ROTL^{30}(T_3)))$
8	T_8	T_7	$ROTL^{30}(T_6)$	$ROTL^{30}(T_5)$	$ROTL^{30}(T_4)$	$ROTL^5(T_7)+K_8+ROTL^{30}(T_3)+W_8+((T_6 \wedge ROTL^{30}(T_5)) \vee (\neg T_6 \wedge ROTL^{30}(T_4)))$
9	T_9	T_8	$ROTL^{30}(T_7)$	$ROTL^{30}(T_6)$	$ROTL^{30}(T_5)$	$ROTL^5(T_8) + K_9 + e_8 + W_9 + ((T_7 \wedge c_8) \vee (\neg T_7 \wedge d_8))$
10	T_{10}	T_9	$ROTL^{30}(T_8)$	$ROTL^{30}(T_7)$	$ROTL^{30}(T_6)$	$ROTL^5(T_9) + K_{10} + e_9 + W_{10} + ((T_8 \wedge c_9) \vee (\neg T_8 \wedge d_9))$
11	T_{11}	T_{10}	$ROTL^{30}(T_9)$	$ROTL^{30}(T_8)$	$ROTL^{30}(T_7)$	$S(T_{10}, 5) + K_{11} + e_{10} + W_{11} + ((T_9 \wedge c_{10}) \vee (\neg T_9 \wedge d_{10}))$

$RAND_0^{10:3}$, and $\{RAND_0^{2:0}||RAND_0^{31:27}\}$ random to recover $T_{4.k}^{18:11}$, $T_{4.k}^{10:3}$, and $\{T_4^{2:0}||T_4^{31:27}\}$, respectively. Finally, we can obtain $T_{4.k}$. If we know $T_{4.k}$, we can compute T_4 with the known W_4. Similarly, if we wish to recover $T_t, t \geqslant 5$, we can perform CPA for $ROTL^5(T_t)$ in RD_{t+1} to recover $T_t.k$ and then compute the corresponding T_t with the known $T_t.p$.

Let us look at Table 3 and consider the unfolded T in each round. Based on the unfolded T_8, if we know T_4, T_5, T_6, T_7, and T_8 (K_8 and W_8 are known), we can recover T_3. Then, based on the unfolded T_7, with the known T_3, T_4, T_5, T_6, T_7, K_7 and W_8, we can recover T_2. Similarly, we can recover T_1, T_0, a, b, c, and d in order from the unfolded T_6 to T_1, respectively. Thus, we need to recover T_4 to T_8 and then calculate T_3, T_2, T_1, T_0, a, b, c, and d in order. For an 8-bit MCU, we need to perform CPA 20 times. Finally, we can obtain *A-key* with the following formulas:

$$\begin{cases} A\text{-}key_0 = a \oplus \text{0x67452301}, \\ A\text{-}key_1 = b \oplus \text{0xEFCDAB89}, \\ A\text{-}key_2 = c \oplus \text{0x98BADCFE}, \\ A\text{-}key_3 = d \oplus \text{0x10325476}. \end{cases}$$

4 Experiments

We implemented the SSD generation procedure and SHA-1 based AKA on an 8-bit MCU, ATMega128A. We chose and controlled only the 56-bit RANDSSD for SSD generation and the 128-bit *RAND* for SHA-1 based AKA; all other parameters were fixed. A 5 Ω resistor was inserted between MCU and GND to monitor the power consumption. We used a LeCroy oscilloscope (WaveRunner 610Zi) to collect the power traces. The sampling rate was 250 M/s. Moreover, we tested a real SIM card that provides cdmaOne and CDMA2000 services. We used a self-made card reader to communicate with the SIM card through the ISO/IES 7816 protocol, and we inserted a suitable resistor to monitor the power consumption. The oscilloscope and the sampling rate were the same as for the MCU. To execute the SSD generation procedure or SHA-1 based AKA in a SIM card, it is necessary to send a specific application protocol data unit (APDU), which is a common command format for ISO/IES 7816. According to the specification[11], the APDUs of SSD generation and SHA-1 based AKA are presented in Fig.9. For our SIM card, SSD generation could be executed successfully. However, we found that there was no SHA-1 based AKA procedure available, meaning that this SIM card is a legacy 3G card and supports only the CAVE-based protocol for 2G and 3G. Therefore, we will report experiments performed on both the MCU and the SIM card for SSD generation but only an experiment performed on the MCU for SHA-1 based AKA.

[11]Removable user identity module for spread spectrum systems, Jul. 2009. https://www.3gpp2.org/Public_html/Specs/C.S0023-D_v1.0_R-UIM-090720.pdf, July 2019.

```
1. APDU for SSD generation:
// Select the main file (3F00).
A0 A4 00 00 02 3F00
// Select the CDMA application (7F25)
A0 A4 00 00 02 7F25
// Base station challenge (11223344)
A0 8A 00 00 04 11223344
// Update SSD.
A0 84 00 00 0F (7-byte RANDSSD) 00 (7-byte ESN)
2. APDU for SHA-1-based AKA:
// Select the main file (3F00).
A0 A4 00 00 02 3F00
// Select the CDMA application (7F25).
A0 A4 00 00 02 7F25
// AKA authentication.
A0 88 01 00 21 (16-byte RAND) 10 (16-byte AUTN)
```

Fig.9. APDUs for SSD generation and SHA-1 based AKA protocol.

4.1 Experiments for SSD Generation

4.1.1 MCU

We chose *RANDSSD*[3] and *RANDSSD*[4] randomly, while the other bytes of *RANDSSD* remained fixed. We acquired 20 000 power traces. Fig.10(a) shows a single power trace for all rounds of CAVE. A zoomed-in view of the first round, showing the 16 iterations, is presented in Fig.10(b).

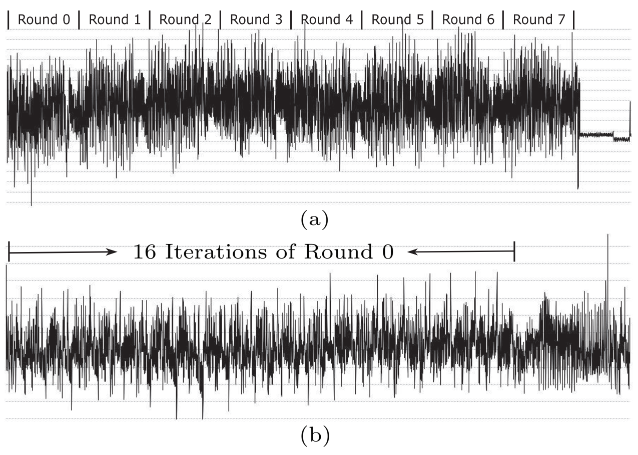


Fig.10. Power traces for (a) 8 rounds of CAVE and (b) the 16 iterations of round 0.

We extracted the traces from the first eight iterations of the first round and analyzed the trace set as described in Subsection 3.2. Fig.11 shows the results

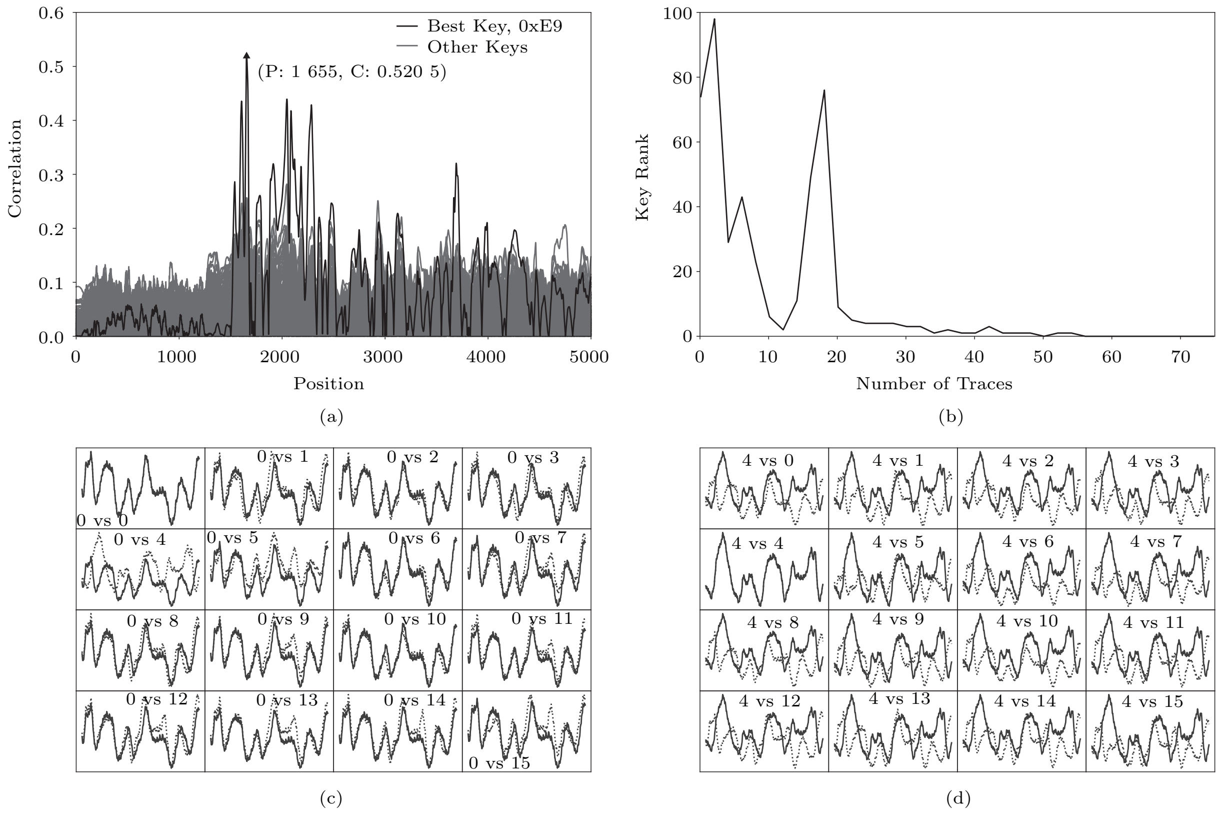
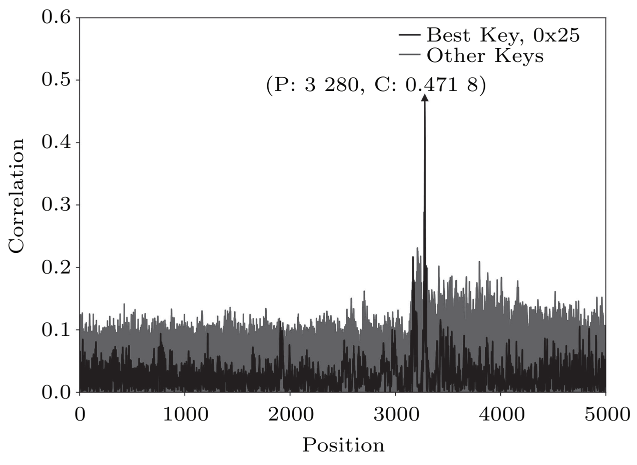


Fig.11. CPA and SPA results for $IR_{0.0}$ on MCU. (a) Correlation traces of 256 candidate keys. The solid triangle represents the highest correlation value (C) and its position (P). (b) Correct key (0xE9) rank vs the number of traces. (c) Mean trace of group 0 (solid line) vs the mean traces of all the 16 groups (dotted line). (d) Mean trace of group 4 (solid line) vs the mean traces of all the 16 groups (dotted line). The numbers 0–15 in (c) and (d) represent the indexes of the groups.

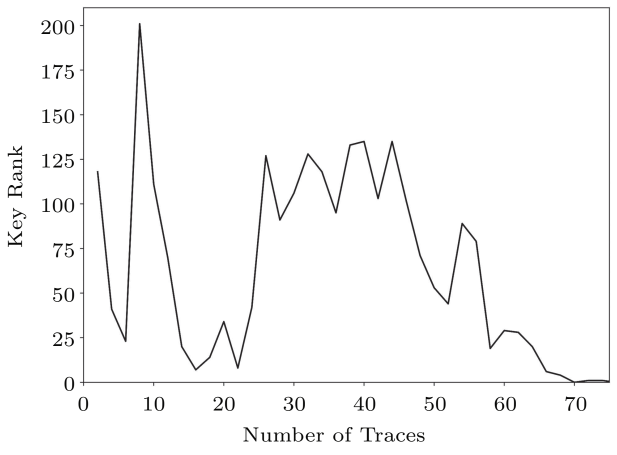
of CPA and SPA for $IR_{0.0}$. Fig.11(a) shows the correlation traces from point 0 to point 5 000 for all possible $IR_{0.0.k}$, from which it can be seen that 0xE9 was the best key with the highest correlation coefficient of 0.520 5, which was far greater than the others. Therefore, 0xE9 was the correct $IR_{0.0.k}$. We summarize the relationship between the number of traces and the rank of 0xE9 in Fig.11(b), from which it can be seen that 60 traces were sufficient for stable key recovery. Then, we used 0xE9 to calculate the correct *lowNibble* for each trace and divided the traces into 16 groups according to their *lowNibble* values. The mean trace of group 0 is plotted for comparison with the mean traces of all 16 groups in Fig.11(c). We can see that all groups were similar except group 4. For confirmation, we present similar plots comparing group 4 with all 16 groups in Fig.11(d). These plots clearly indicate that group 4 was the distinct one; therefore, $IR_{0.0.n}$ was 4. Group 4, which contains 1 258 traces, was then filtered out, leaving 18 742 traces for subsequent analysis. In the final step ($IR_{7.1}$), there were 7 340 traces remaining, which were sufficient to complete the analysis.

4.1.2 *SIM Card*

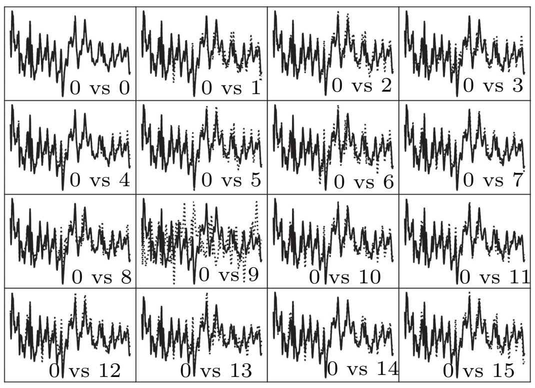
As in the case of the MCU, we collected 20 000 power traces, located the first eight iterations of the first round, and then applied the strategy to this trace set. Fig.12 shows the results for $IR_{0.0}$. These summarized results are similar to those in Fig.11 and prove that the leakage from the SIM card is almost the same as the leakage from the 8-bit MCU we used. For successful CPA, fewer than 100 traces were needed. $IR_{0.0.k}$ was 0x25, and $IR_{0.0.n}$ was 9. All analyses could be completed within 10 minutes.



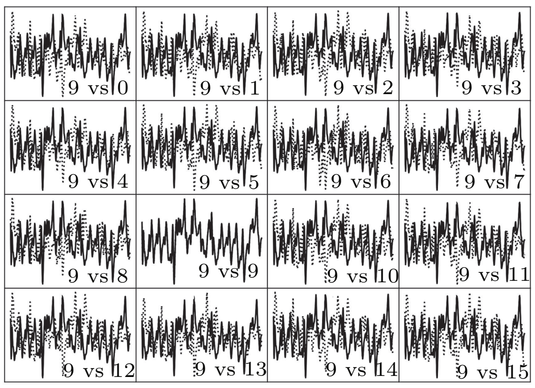
(a)



(b)



(c)



(d)

Fig.12. CPA and SPA results for $IR_{0.0}$ on SIM card. (a) Correlation traces of 256 candidate keys. The solid triangle represents the highest correlation value (C) and its position (P). (b) Correct key (0x25) rank vs the number of traces. (c) Mean trace of group 0 (solid line) vs the mean traces of all the 16 groups (dotted line). (d) Mean trace of group 4 (solid line) vs the mean traces of all the 16 groups (dotted line). The numbers 0–15 in (c) and (d) represent the indexes of the groups.

4.2 Experiment for SHA-1 Based AKA

For SHA-1 based AKA, since the experiment was performed on the 8-bit MCU, we performed CPA in a byte-by-byte manner. As described in Subsection 3.3, we first needed to infer T_4. We acquired the power traces related to $Round_5$ and $Round_6$ with the chosen W_4, as shown in Fig.13. We recovered $T_{4.k}^{26:19}$, $T_{4.k}^{18:11}$, and $T_{4.k}^{10:3}$, $\{T_4^{2:0}||T_4^{31:27}\}$ within 2 000 traces. The results are shown in Fig.14. Fig.14(a) shows the correlation traces of all candidates for $T_{4.k}^{26:19}$; the best candidate key was 0x7E. Fig.14(b) shows the correlation traces of all candidates for $T_{4.k}^{18:11}$; the best candidate key was 0x70. Fig.14(c) shows the correlation traces of all candidates for $T_{4.k}^{10:3}$; the best candidate key was 0x24. Fig.14(d) shows the correlation traces of all candidates for $\{T_4^{2:0}||T_4^{31:27}\}$; the best candidate key was 0x48. Therefore, $T_{4.k}$ was 0x4BF38120. Then, we calculated T_4 with the known W_4 and performed CPA for $T_{5.k}$. Similarly, we could obtain $T_{5.k}$, $T_{6.k}$, $T_{7.k}$, and $T_{8.k}$. Then, we derived T_3, T_2, T_1, a, b, c, and d from T_4, T_5, T_6, T_7, and T_8. Finally, we obtained *A-key*. Each round of CPA required no more than 2 000 traces, and the total number of traces we needed to acquire was approximately 40 000.

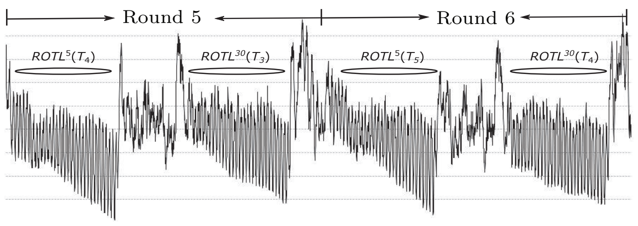


Fig.13. Power traces for the round 5 and the round 6 of SHA-1.

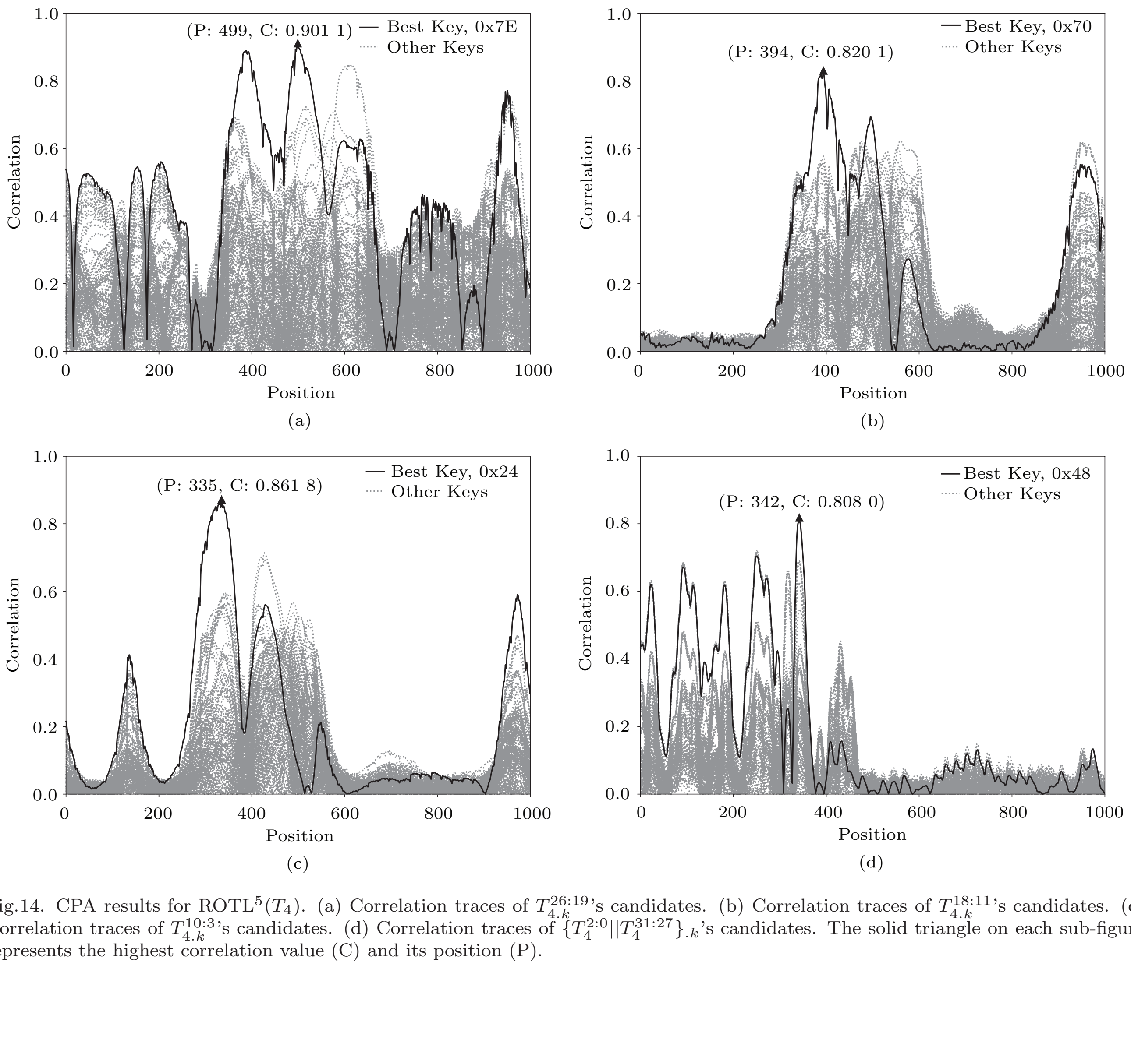


Fig.14. CPA results for $\mathrm{ROTL}^5(T_4)$. (a) Correlation traces of $T_{4.k}^{26:19}$'s candidates. (b) Correlation traces of $T_{4.k}^{18:11}$'s candidates. (c) Correlation traces of $T_{4.k}^{10:3}$'s candidates. (d) Correlation traces of $\{T_4^{2:0}||T_4^{31:27}\}_{.k}$'s candidates. The solid triangle on each sub-figure represents the highest correlation value (C) and its position (P).

5 Conclusions

This paper reports the side-channel weaknesses of the two main authentication protocols of CDMA networks. These weaknesses are vulnerable to the CPA and the SPA, which cause that the secret internal data of the protocols can be recovered. Based on the weaknesses, we proposed two specific attack strategies to recover the whole authentication key for the two protocols, respectively. Through these strategies, the authentication keys of the protocols can be extracted easily, which could cause a series of security problems. The attack strategies are verified both on an 8-bit MCU and a real SIM card. The experimental results showed that our strategies are effective and demonstrated that the authentication protocols of CDMA networks are not secure against SCA.

In the future, it will be interesting to develop the lightweight countermeasures for the CAVE-based and the AKA-based protocols. On the other hand, 5G technology is under development, and it will appear on the market soon. The physical security of its authentication protocol is still important. Thus, the research of how to protect the SIM cards of the new generation against SCA is very meaningful. We leave these topics for future study.

References

[1] Shankar P M. Introduction to Wireless Systems. Wiley, 2002.

[2] Sauter M. From GSM to LTE: An Introduction to Mobile Networks and Mobile Broadband (1st edition). Wiley, 2011.

[3] Bertoni G, Daemen J, Peeters M, van Assche G. Keccak. In *Proc. the 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques*, May 2013, pp.313-314.

[4] Steele R, Lee C C, Gould P. GSM, cdmaOne and 3G Systems (1st edition). Wiley, 2001.

[5] Mangard S, Oswald E, Popp T. Power Analysis Attacks: Revealing the Secrets of Smart Cards. Springer, 2007.

[6] Rao J R, Rohatgi P, Scherzer H, Tinguely S. Partitioning attacks: Or how to rapidly clone some GSM cards. In *Proc. the 2002 IEEE Symposium on Security and Privacy*, May 2002, pp.31-41.

[7] Zhou Y, Yu Y, Standaert F X, Quisquater J J. On the need of physical security for small embedded devices: A case study with COMP128-1 implementations in SIM cards. In *Proc. the 17th International Conference on Financial Cryptography and Data Security*, April 2013, pp.230-238.

[8] Liu J, Yu Y, Standaert F X, Guo Z, Gu D, Sun W, Ge Y, Xie X. Small tweaks do not help: Differential power analysis of MILENAGE implementations in 3G/4G USIM cards. In *Proc. the 20th European Symposium on Research in Computer Security*, September 2015, pp.468-480.

[9] Maghrebi H, Bringer J. Side-channel analysis of the TUAK algorithm used for authentication and key agreement in 3G/4G networks. In *Proc. the 15th International Conference on Smart Card Research and Advanced Applications*, November 2016, pp.39-56.

[10] Brier E, Clavier C, Olivier F. Correlation power analysis with a leakage model. In *Proc. the 6th International Workshop on Cryptographic Hardware and Embedded Systems*, August 2004, pp.16-29.

[11] Kocher P, Jaffe J, Jun B. Differential power analysis. In *Proc. the 19th Annual International Cryptology Conference*, August 1999, pp.388-397.

Chi Zhang received his B.S. degree in computer science and technology from Southeast University, Nanjing, in 2014. He is currently a Ph.D. candidate at School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai. His research interests include hardware security, signal processing, and machine learning for side-channel analysis.

Jun-Rong Liu received his B.S. degree in applied mathematics from Xidian University of China, Xi'an, in 1992, M.S. degree in computer application technology from Beijing University of Posts and Telecommunications, Beijing, in 2001, and Ph.D. degree in computer science and engineering from Shanghai Jiao Tong University, Shanghai, in 2016. He is currently working at ZhiXun Crypto Testing and Evaluation Technology Co., Ltd, Shanghai. His research interests include cryptography, side-channel analysis, and hardware security.

Da-Wu Gu is a distinguished professor at School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University (SJTU), Shanghai. He received from Xidian University of China, Xi'an, his B.S. degree in applied mathematics in 1992, and his M.S. degree in 1995 and Ph.D. degree in 1998 both in cryptography. His research interests include crypto algorithms, crypto engineering, and system security. He leads the Laboratory of Cryptology and Computer Security (LoCCS) at SJTU, Shanghai. He was the winner of Chang Jiang Scholars Distinguished Professors Program in 2014 by Ministry of Education of China. He won the National Award of Science and Technology Progress in 2017. He has got over 150 scientific papers in academic journals and conferences, and owned 28 innovation patents.

Wei-Jia Wang received his Ph.D. degree in computer science and engineering from Shanghai Jiao Tong University, Shanghai, in 2018. He is currently a post-doctoral researcher at Crypto Group, Electrical Engineering Department (ELEN), Institute of Information and Communication Technologies (ICTEAM), Cathdic University of Louvain, Louvain-la-Neuve. His research interests include side-channel analysis, leakage resilient, cryptographic implementations, and hardware security.

Xiang-Jun Lu received his B.S. degree in software engineering from Northwestern Polytechnical University, Xi'an, in 2015. He is currently a Ph.D. candidate at the School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai. His research interests include side-channel analysis, hardware security, machine learning, and neural networks.

Zheng Guo received his B.S degree in electronic engineering, his M.S. degree in communication and information system, and his Ph.D. degree in computer science from Shanghai Jiao Tong University, Shanghai, in 2002, 2005, and 2016, respectively. He is currently working at ZhiXun Crypto Testing and Evaluation Technology Co., Ltd, Shanghai. His research interests include side-channel analysis and IC design.

Hai-Ning Lu is a lecturer at School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai. He received his B.S. degree in 2001 and his M.S. degree in 2004, both in computer science and engineering, from Shanghai Jiao Tong University, Shanghai. He is currently working at Shanghai Viewsource Information Science and Technology Co., Ltd, Shanghai. His research interests include cryptography based cloud security and network security.