Side-Channel Analysis of the TUAK Algorithm used for Authentication and Key Agreement in 3G/4G Networks*

Houssem Maghrebi, Julien Bringer

SAFRAN Identity & Security
firstname.lastname@morpho.com

Abstract. Side-channel attacks are nowadays well known and most designers of security embedded systems are aware of them. Yet, these attacks are still major concerns and several implementations of cryptographic algorithms are still being broken. In fact, a recent work has exhibited a successful Differential Power Attack (DPA) on the MILENAGE algorithm used for authentication and key agreement in UMTS/LTE networks. Surprisingly, the targeted MILENAGE implementations in different USIM cards, coming from several mobile network operators, didn't systematically take advantage of the large panel of the well-known side-channel countermeasures. Recently, a new algorithm called TUAK, based on the KECCAK permutation function, has been proposed as alternative to MILENAGE. Although KECCAK was deeply analyzed in several works, the TUAK algorithm needs to be well investigated to assess its security level and to avoid inappropriate apply of KECCAK. In this paper, we present a side-channel analysis of an unprotected TUAK implementation and we demonstrate that a successful side-channel attack is possible if the state-of-the-art countermeasures are not considered. Our results show that a few hundred of traces would roughly be needed to recover the subscriber key and other authentication secrets fixed by mobile operators. Actually, this work raises a warning flag to embedded systems developers alerting them to rely on adequate countermeasures, which effect shall be confirmed with thorough security analysis, when implementing cryptographic primitives in USIM cards.

Keywords: TUAK, KECCAK, side-channel analysis, authentication and key agreement, UMTS, LTE, USIM cards.

1 Introduction

Side-Channel Attacks. Side-channel attacks (SCA) are a serious threat against modern cryptographic implementations. They exploit information leaked from the physical implementations of cryptographic algorithms. Since this leakage (*e.g.* the power consumption or the electromagnetic radiation) depends on the

* This work has been partially funded by the ANR project SERTIF.

internally used secret key, the adversary may perform an efficient key-recovery attack to reveal this sensitive data. During the two last decades, the development of the smart card industry has urged the cryptographic research community to carry on with this new concept of attacks and many papers describing either countermeasures or attacks improvement have been published. Amongst the side-channel attacks, two classes may be distinguished. The set of so-called *profiling SCA*: is the most powerful kind of side-channel attacks and consists of two steps. First, the adversary procures a copy of the *target device* and uses it to characterize the physical leakage. Second, she performs a key-recovery attack on the target device. This set of profiled attacks includes Template attacks [15] and Stochastic models (*aka* Linear Regression Analysis) [17,30,31]. The set of so-called *non-profiling SCA*: corresponds to a much weaker adversary who has only access to the physical leakage captured on the target device. To recover the internally used secret key, she performs some statistical analyses to detect the dependency between the leakage measurements and this sensitive variable. This set of non-profiled attacks includes Differential Power Analysis (DPA) [22], Correlation Power Analysis (CPA) [13] and Mutual Information Analysis (MIA) [18].

On the Vulnerability of Cryptographic Algorithms used in Cellular Networks. Till recent times, research towards the security of cryptographic algorithms used in *mobile cellular networks* has been essentially concerned with classical cryptanalytic attacks [8,34]. Even if the cryptographic algorithm is mathematically sound [7], an adversary can mount a side-channel attack on a Universal Subscriber Identity Module (USIM) card implementing it to recover the internally used secret key. Indeed, authors in [28] have described a successful side-channel attack on the popular GSM authentication algorithm COMP128. Mainly, the entire 128-bit key of COMP128 can be recovered with only 8 adaptively chosen inputs.

Recently, Liu *et al.* have assessed in [24] a physical security analysis of the MILENAGE algorithm [4]. This primitive is used for authentication and key agreement in UMTS (Universal Mobile Telecommunications System)/LTE (Long-Term Evolution) networks and it is based on the AES algorithm [16]. The authors have concluded that the targeted MILENAGE implementations in several USIM cards didn't systematically take advantage of state-of-the-art countermeasures against side-channel attacks. In fact, when performing a DPA attack, an adversary is able to recover the subscriber key and other secrets fixed by the mobile network operator within a few hundred of traces for the unprotected USIM cards which have been analyzed. The results of this security analysis have been also presented at BlackHat 2015 [23].

Our Contribution. The 3$^{\text{rd}}$ Generation Partnership Project (3GPP) has proposed the TUAK algorithm as an alternative to MILENAGE to enhance security and privacy of UMTS/LTE mobile cellular networks [5]. The design of TUAK is based upon the winner of the SHA-3 competition, KECCAK [11]. When looking carefully at the technical specification of the TUAK algorithm [5, Sec. 7.3], one

can clearly notice that it is highly recommended to protect its implementation against side-channel attacks, as it was the case for MILENAGE [4, Sec. 5.4]. In this paper, we try to answer the following question:

Is the straightforward TUAK *implementation, in a USIM card, protected against side-channel attacks?*

Actually, we answer this question negatively. In fact, our conducted security analysis has led to the same conclusions as those obtained in [24]. Mainly, we show first that it is possible to recover the involved secret parameters with roughly a few hundred of TUAK invocations. Then, we try to bridge the gap between the physical implementation of this new cryptographic primitive in USIM cards and the existing investigations on dedicated countermeasures to thwart side-channel attacks by proposing some defensive strategies.

Our paper provides, to the best of our knowledge, the first complete side-channel analysis of the TUAK algorithm. The only publicly available security analysis was performed in [26], where authors have theoretically evaluated the robustness of the TUAK implementation against timing attacks [21].

Paper Outline. The paper is organized as follows. In Sec. 2, we first overview the AKA protocol used for mutual authentication and key agreement in 3G/4G mobile cellular networks. Then, we describe the main characteristics of the TUAK algorithm and the KECCAK permutation function. In Sec. 3, we detail our SCA strategy and provide the practical results of our proposal. This is followed in Sec. 4 by a discussion of some well-known side-channel countermeasures and other defensive strategies that could be applied to protect the TUAK algorithm when implemented in USIM cards. Finally, Sec. 5 draws general conclusions.

2 Background

2.1 Overview of the AKA Protocol

The 3GPP is a collaboration agreement that was established in 1998 between international telecommunications standards bodies. It aims, inter alia, at the development of technical specifications for the third and the fourth generation (3G/4G) mobile cellular networks. To enhance the security of the authentication procedure through these new generation networks, the 3GPP has proposed a mutual Authentication and Key Agreement (AKA) protocol [1]. In Fig. 1, we describe the sequence diagram of this protocol in a unified way for 3G and 4G networks[1]. Without loss of generality, the 3G/4G networks are composed of:

- The user equipment: comprises the mobile equipment (*e.g.* a cellular phone) and the USIM card issued by a mobile operator to a subscriber and that contains mainly: the International Mobile Subscriber Identity (IMSI), which is the permanent key of the subscription, and the cryptographic keys and algorithms required for the authentication protocol.

[1] The AKA 4G protocol slightly differs from the 3G one, in particular on the way that the session keys are computed.

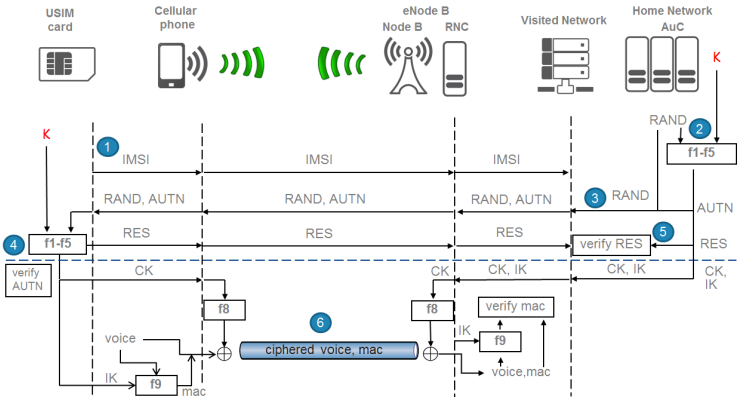

Fig. 1. Unified AKA protocol between the USIM card and the authentication center in 3G/4G networks.

- The radio access network: which contains the Node B's and the Radio Network Controllers (RNC) for the 3G network and is composed only by the Evolved Node B's for the 4G network. It provides and manages wireless connection between mobile equipment and the core network.
- The visited network: it acts as the manager of network connectivity and is responsible for the subscriber authentication.
- The home network: which contains and maintains the Authentication Center (AuC) server. The latter contains the keys of all operator's subscribers and generates temporary tokens used to establish authentication, to set up session keys and to enable the communication.

The AKA protocol is executed between the user, the visited network and the home network as follows: when the mobile equipment is powered on, it sends through wireless communication the IMSI to the visited network. The latter forwards the request to the AuC. Upon receipt of this request, the AuC generates a fresh nonce (RAND) and fetches the subscriber key K to generate a batch of authentication vectors by computing the cryptographic functions (f_1, f_2, f_3, f_4, f_5). Each authentication vector is composed of:

- RES: the expected response value,
- CK, IK: respectively the cipher and the integrity session keys, and
- AUTN: the authentication token which contains several concatenated fields such that:

$$\text{AUTN} = (\text{SQN} \oplus \text{AK}) \| \text{AMF} \| \text{MAC} \ , \tag{1}$$

where SQN is the sequence number, AK is the anonymity key, AMF is the authentication management field and MAC is the message authentication code. The ($\|$) and ($\oplus$) symbols denote respectively the concatenation and the exclusive-or (XOR) operations. Then, the AuC sends the AUTN and the challenge (RAND)

to the user equipment and keeps the other related security parameters (RES, CK and IK). Once receipt of (AUTN, RAND), the USIM card executes symmetrically the same security functions (f_1, f_2, f_3, f_4, f_5) to check the parameters of the authentication token AUTN. First, it computes the anonymity key AK (by performing f_5 function) and retrieves the SQN = (SQN $\oplus$ AK) $\oplus$ AK. Second, it computes the MAC (by executing f_1 function) and then compares it to that contained in AUTN. If they are equal, then the USIM card performs a check on the SQN sent by the AuC and that maintained inside the USIM card. If either of these two checks fails, then the USIM card sends some error messages. Otherwise, it generates the session keys (CK, IK), computes the response RES and sends it to the visited network. Next, the visited network compares the response received from the AuC and the one received from the USIM card. If the check succeeds, then the AKA protocol is completed. Finally, the subscriber could start his communication. His voice is encrypted and integrity protected (with a MAC) by the mobile (and the network) using the security functions (f_8, f_9) and the generated session keys (CK, IK). For further details on how these ciphering and integrity functions are used, we refer the reader to [2,3].

The AKA protocol is compatible with either MILENAGE or TUAK algorithms. The main difference is the security functions used (f_1, f_2, f_3, f_4, f_5). In fact, while MILENAGE security functions are based on the AES-128, the TUAK ones are based on the KECCAK permutation.

2.2 The TUAK Algorithm

The TUAK algorithm is composed of several encryption functions that are based on the KECCAK permutation. Mainly, it contains 8 different functions:

- the derivation function: used to compute a 256-bit value, called TOP_c, from the TOP, a secret constant value chosen by the mobile operator, and the subscriber key K. As stated in [5, Sec. 7.1], this function is preferably performed off-card and only the output (*aka* TOP_c) is stored in the USIM card. For the rest of this paper, we will consider this recommendation as part of our assumptions.
- f_1: used to generate the message authentication code (MAC).
- f_2: used to generate the response (RES).
- f_3: used to generate the cipher session key (CK).
- f_4: used to generate the integrity session key (IK).
- f_5: used to generate the anonymity key (AK).
- f_1^* and f_5^*: used to recompute MAC and AK in case of authentication failure (*aka* re-synchronization phase) to determine the error messages to send.

All the previously described functions take 1600-bit as input. However, the output length varies according to each function.

Without loss of generality, we will focus in this work on the implementation of the f_1 function on the USIM card side. We stress the fact that our attack strategy could be applied to any other TUAK functions. However, in realistic

scenarios if the MAC comparison fails, the authentication is aborted and the rest of TUAK functions (*i.e.* f_2, f_3 and f_4) will not be processed. Moreover, we argue that our choice differs from that in [24], where the authors have focused on f_5 and justified their choice by the fact that this function is the first one to be executed. But, when looking at the technical specification, we noticed that during the establishment of the AKA protocol the use of f_5 function is optional [1, Sec. 5.1.6.7] which further supports our choice towards targeting the f_1 function. In fact, most of mobile network designers omit this calculation aiming at enhancing the performance of the authentication process. Indeed, in such a case, the SQN field is sent in clear (*i.e.* not XORed with the anonymity key AK as detailed in Eq. (1) and hence, AUTN = SQN$\|$AMF$\|$MAC). We will use this technical detail when exhibiting our attack strategy.

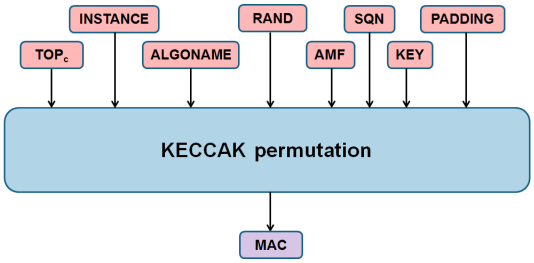

Fig. 2. The f_1 function of TUAK.

As depicted in Fig. 2, f_1 function takes as input:

- TOP$_c$: a 256-bit secret value.
- INSTANCE: an 8-bit constant value that depends on the subscriber key length (128-bit or 256-bit) and the outputted MAC length (64-bit, 128-bit or 256-bit). In the sequel, we fix the INSTANCE field at the hexadecimal value 0x10. Said differently, we assume a subscriber key of 128-bit length and an outputted MAC of 64-bit length[2].
- ALGONAME: a 56-bit constant value equals to the ASCII representation of the string "TUAK1.0"[3].
- RAND: a 128-bit value standing for the nonce generated by the AuC.
- AMF: a 16-bit value standing for the authentication management field.
- SQN: a 48-bit value standing for the sequence number.
- KEY: the subscriber key of 128 or 256 bits length. As explained above, we will fix the key length at 128-bit.

[2] We refer the reader to [5, Sec. 6.2] for further information about the specifications of the INSTANCE field.

[3] Explicitly, the successive ALGONAME field bytes are: 0x30, 0x2E, 0x31, 0x4B, 0x41, 0x55 and 0x54.

– PADDING: an 832-bit value to reach the 1600-bit required size for the input.

Yet, we notice that for the other TUAK functions, the same inputs are used except for the INSTANCE field which changes according to the required output to be calculated (*e.g.* CK, IK, RES).

To compute the MAC, f_1 function performs the KECCAK permutation on the 1600-bit input described above. In the following subsection, we will detail the KECCAK permutation.

2.3 The KECCAK Permutation

In 2012, the NIST selected KECCAK as the winner of the SHA-3 contest due to its novel design and excellent performance both in resistance to various attacks and implementation in software and hardware. The structure of KECCAK [11] is based on the so-called *sponge function* which is a generalized hash function that accepts an arbitrary length inputs (of the message and key) and generates digests of any desired bit size. Thus, KECCAK could be used in several ways: to perform a regular hashing, to compute a MAC, to derive keys, *etc.*

Concerning the internal KECCAK permutation function, there are 7 instances with an input width ranging from 25 to 1600 bits increasing in powers of two (*i.e.* the permutation width is equal to 25×2^l, *s.t.* $l \in [0, 6]$). The default input width value is 1600 bits [11], as it is the case for the TUAK security functions [5].

Besides, several bit-width implementations of the KECCAK permutation could be considered. For instance, in [5, Annex. *E*], the 3GPP technical specification provides examples of source code for an 8, 32 and 64 bits implementation of the KECCAK permutation. In the sequel, we will focus on the 32-bit version since it especially suitable for the USIM card we targeted for our side-channel analysis. Doing so, the 1600-bit state of the KECCAK permutation (*i.e.* the input of the TUAK f_1 function) are represented as 50 blocks of 32 bits each. We recall the source code of the 32-bit implementation, that we will use, in Listing. 1.

From Listing. 1, one can notice that the KECCAK permutation consists of 24 rounds, each round is composed of 5 functions:

– θ function: consists of bitwise XOR operations of the input.
– ρ function: consists of bitwise rotation of the input.
– π function: is a simple permutation over the bits of the state.
– χ function: is the unique non-linear function of the KECCAK permutation. It consists of a mix between XOR, AND and NOT binary operations.
– ι function: is a bitwise XOR with a round constant.

We recall that these functions operate on states of 32-bit each, *i.e.* $s[i]$ in Listing. 1. According to this implementation version, we describe in Fig. 3 how the 1600-bit inputs of the TUAK f_1 function are represented. So, each line corresponds to a state of 32-bit.

```
void KECCAK_f_32(uint32 s[50])
{
   uint32 t[10];
   uint8 i, j, round, k;

   for(round=0; round<24; ++round)
   {
      /* Theta function */
      for(i=0; i<10; i=i+2)
      {
         t[i] = s[i] ^ s[10+i] ^ s[20+i] ^ s[30+i] ^ s[40+i];
         t[i+1]=s[i+1]^s[10+i+1]^s[20+i+1]^s[30+i+1]^s[40+i+1];
      }

      for(i = 0; i < 5; ++i, s+=10)
      {
         s[0] ^= t[8] ^ ((t[2]<<1)|(t[3]>>31));
         s[1] ^= t[9] ^ ((t[3]<<1)|(t[2]>>31));
         s[2] ^= t[0] ^ ((t[4]<<1)|(t[5]>>31));
         s[3] ^= t[1] ^ ((t[5]<<1)|(t[4]>>31));
         s[4] ^= t[2] ^ ((t[6]<<1)|(t[7]>>31));
         s[5] ^= t[3] ^ ((t[7]<<1)|(t[6]>>31));
         s[6] ^= t[4] ^ ((t[8]<<1)|(t[9]>>31));
         s[7] ^= t[5] ^ ((t[9]<<1)|(t[8]>>31));
         s[8] ^= t[6] ^ ((t[0]<<1)|(t[1]>>31));
         s[9] ^= t[7] ^ ((t[1]<<1)|(t[0]>>31));
      }
      s -= 50;

      /* Rho function */
      for(i=2; i<50; i+=2)
      {  k = Rho[i>>1] & 0x1f;
         t[0] = (s[i+1] << k) | (s[i] >> (32-k));
         t[1] = (s[i] << k) | (s[i+1] >> (32-k));
         k = Rho[i>>1] >> 5;
         s[i] = t[1-k], s[i+1] = t[k];
      }

      /* Pi function */
      for(i=2, t[0]=s[2], t[1]=s[3]; (j=(Pi[i>>1]<<1))>2; i=j)
      {
         s[i]=s[j], s[i+1]=s[j+1];
         s[i]=t[0], s[i+1]=t[1];
      }

      /* Chi function */
      for(i=0; i<5; ++i, s+=10)
      {
         for(j=0; j<10; ++j)
            t[j] = (~s[(j+2)%10]) & s[(j+4)%10];
         for(j=0; j<10; ++j)
            s[j] ^= t[j];
      }
      s -= 50;

      /* Iota function */
      t[0] = Iota[round];
      s[0] ^= (t[0] | (t[0]<<11) | (t[0]<<26)) & 0x8000808B;
      s[1] ^= (t[0]<<25) & 0x80000000;
   }
}
```

Listing 1. Source code of the 32-bit version of the KECCAK permutation function, inspired from [5] and [12].

s[0]	TOP$_U$
s[1]	
s[2]	
s[3]	
s[4]	
s[5]	
s[6]	
s[7]	
s[8]	INSTANCE (8 bits) + ALGONAME (24 bits)
s[9]	ALGONAME (32 bits)
s[10]	RAND
s[11]	
s[12]	
s[13]	
s[14]	AMF (16 bits) + SQN (16 bits)
s[15]	SQN(32 bits)
s[16]	KEY
s[17]	
s[18]	
s[19]	
s[20]	0x00000000
s[21]	0x00000000
s[22]	0x00000000
s[23]	0x00000000
s[24]	0xF8000000
s[25]	0x00000000
s[26]	0x00000000
s[27]	0x00000000
s[28]	0x00000000
s[29]	0x00000000
s[30]	0x00000000
s[31]	0x00000000
s[32]	0x00000000
s[33]	0x80000000
s[34]	0x00000000
s[35]	0x00000000
s[36]	0x00000000
s[37]	0x00000000
s[38]	0x00000000
s[39]	0x00000000
s[40]	0x00000000
s[41]	0x00000000
s[42]	0x00000000
s[43]	0x00000000
s[44]	0x00000000
s[45]	0x00000000
s[46]	0x00000000
s[47]	0x00000000
s[48]	0x00000000
s[49]	0x00000000

Fig. 3. The inputs of the TUAK f_1 function.

3 Side-Channel Analysis of the TUAK Implementation in USIM Cards

Despite KECCAK was carefully studied by the researchers [10,25,32,35] to assess its security level against side-channel attacks, the TUAK algorithm need to be well investigated to avoid inappropriate application KECCAK such that the security properties of the latter are decreased. We propose in this section a side-channel analysis of the TUAK algorithm and we first start by describing our attack strategy and the identified attack paths.

3.1 Side-Channel Analysis Strategy

As detailed in [24], the ultimate purpose of the adversary is to recover the subscriber key and the TOP_c stored in the USIM card, and then to create a cloned one by personalizing a blank USIM card with these secret values. In order to recover the key and the TOP_c by means of SCA, we identify in what follows the sensitive operations of the TUAK f_1 function that could leak information about the internally used secret parameters. Mainly, we will focus on the execution of the first KECCAK permutation round and especially on the θ function.

Before detailing our attack strategy, let us first recall the different assumptions considered throughout the previous sections. First, the TOP_c is computed off-card and stored in the USIM card. Second, the subscriber key length is 128-bit. Third, the TUAK f_5 function is omitted since optional. Thus, the SQN is known to the adversary.

Under the previous assumptions, we provide hereafter an in-depth description of our attack proposal.

Recovering the TOP_c. When looking carefully at the loop in Lines $(9-13)$ of Listing. 1, one can notice that some temporary states are computed by XORing the KECCAK permutation input blocks (s table). For instance, when the loop index i equals 0, the temporary state $T[0]$ satisfies:

$$T[0] = s[0] \oplus s[10] \oplus s[20] \oplus s[30] \oplus s[40] \ , \tag{2}$$

where $s[i]$ is the i^{th} 32-bit block input of the TUAK f_1 function. Now, when focusing on Fig. 3, one can identify the input parameters standing for each $s[i]$ block of Eq. (2). In fact, one can conclude that:

- $s[0]$: stands for the first 32-bit block of the TOP_c.
- $s[10]$: stands for the first 32-bit block of RAND. We recall that the latter value is known to the adversary.
- $s[20]$, $s[30]$ and $s[40]$: stands each for a 32-bit block of zero-padding.

Consequently, the temporary state $T[0]$ contains the result of the bitwise XOR of the first 32-bit blocks of the TOP_c and the RAND. Hence, a *divide-and-conquer* side-channel attack could be performed to recover separately the first 4 bytes of the TOP_c ($s[0]$).

The same strategy is adapted to retrieve the second, third and fourth blocks of the secret variable TOP_c ($s[1]$, $s[2]$ and $s[3]$). For instance, the adversary can perform a side-channel when targeting the computation of the temporary states $T[1]$, $T[2]$ and $T[3]$ respectively. Besides, all the latter computation involves a XOR of 32-blocks of both TOP_c and RAND.

Now, we will describe how to recover the fifth and the sixth 32-bit blocks of the TOP_c. To do so, we have to look at the loop in Lines $(15-27)$ of Listing. 1 when the loop index i equals 0. At Lines 21 and 22, the fifth ($s[4]$) and the sixth ($s[5]$) blocks of the TOP_c are XORed with the temporary states $T[2]$ and $T[3]$ respectively. The latter values are known to the adversary if she has succeeded his side-channel attack as described above ($s[2]$ and [3] have been recovered). In fact, $T[2]$, respectively $T[3]$, contains the XOR results of the third, receptively the fourth, blocks of TOP_c and the RAND. All in all, the adversary could perform a side-channel attack when targeting the XOR operations in Lines 21 and 22 to carry out the fifth and the sixth blocks of the TOP_c.

Given that $T[4]$ and $T[5]$ are already known[4], the attacker could perform as well a successful side-channel attack when targeting the XOR operations at Lines 23 and 24 when the loop index i equals 0 to guess the last two 32-blocks of TOP_c ($s[6]$ and $s[7]$).

Hence, the attacker recovers the 256-bit value of the secret variable TOP_c. We stress the fact that other XOR operations could be targeted to carry out some blocks of the TOP_c. For instance, to find the fifth and the sixth 32-bit blocks of the TOP_c, one could directly target the computation of temporary states $T[4]$ and $T[5]$. There are, in fact, several possibilities to mount the attack on the TOP_c, depending on the targeted operations the adversary chooses.

Recovering the subscriber key. To recover the 16-byte values of the key, the attacker should target the XOR operations performed at Lines 17, 18, 23 and 24 when the loop index i equals 1.

In fact, at Line 17 when computing $s[10] \oplus T[8]$, we are actually performing a 32-bit XOR operation of: the first 32-bit block of RAND ($s[10]$), the third block of the key ($s[18]$) and the INSTANCE byte concatenated to the first three bytes of the ALGONAME. The latter 4-byte value ($s[8]$) is constant and known to the attacker (*i.e.* the successive bytes values are 0x10, 0x30 0x2E 0x31).

Alike, at Line 18 when computing $s[11] \oplus T[9]$, we are executing a 32-bit XOR operation of: the second 32-bit block of RAND ($s[11]$),the last 4 bytes of the ALGONAME ($s[9]$, *i.e.* the successive bytes values are 0x4B, 0x41 0x55 0x54) and the fourth block of the key ($s[19]$).

Hence, a side-channel attack when targeting the above detailed XOR operations should recover the last 8 bytes of the subscriber key ($s[18]$ and $s[19]$).

[4] In fact, both $T[4]$ and $T[5]$ depend on the SQN and AMF fields which are sent in clear, so known to the adversary, since we have assumed that f_5 function is not executed. Indeed, these fields vary from an authentication session to another one (*e.g.* the SQN is incremented by one for each authentication request) which enables performing a side-channel attack.

Finally, since $T[4]$ and $T[5]$ are known, the first 8 bytes of the subscriber key ($s[16]$ and $s[17]$) can also be recovered with a divide-and-conquer side-channel attack when targeting the XOR operations at Lines 23 and 24.

Doing so, the adversary succeeds to recover all the needed secret data enabling him to clone the USIM card and use it illegally.

3.2 Evaluation Methodology and Setup

To perform our practical analysis, we have first implemented the 32-bit version of the TUAK f_1 function, described in Listing. 1, in a 32-bit processor core running at up to 25 MHz. Then, the USIM card is inserted into a self-made card reader (with a resistor integrated to measure the voltage drop and enable the power acquisition) that has been connected to a monitoring PC via an USB link. In our setup, the PC is playing the role of the AuC. So, it sends the authentication token AUTN and the RAND to the USIM card to initiate the execution of the implemented TUAK f_1 function. While processing the messages, the power consumption traces of the USIM card were collected on a Lecroy WavePro 725Zi oscilloscope with 1.5 GHz bandwidth and a maximal sampling rate of 40 GSa/s. We provide in Fig. 4 a high-level description of our measurement setup.

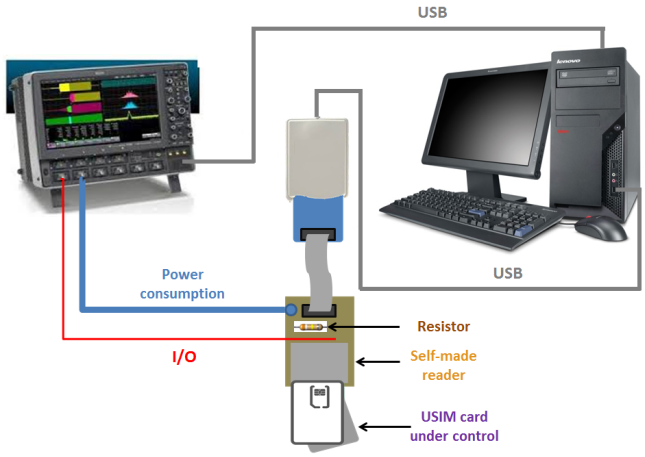

Fig. 4. The measurement setup used for our analysis.

Actually, our side-channel evaluation consists of two steps. First, we will perform a Simple Power Analysis (SPA) to identify the executed operations during the TUAK f_1 function and especially the target area for the attack (*i.e.* the θ function of KECCAK). Second, we will perform a statistical analysis via a Correlation Power attack (CPA) [13], to evaluate whether it is possible to recover the secret data used.

For illustration, the yellow curve at the top of Fig. 5 gives a big view of a captured power trace[5] where we clearly identify the 24 regular rounds of the KECCAK permutation. A zoom on the beginning of the first round is provided at the bottom of Fig. 5.

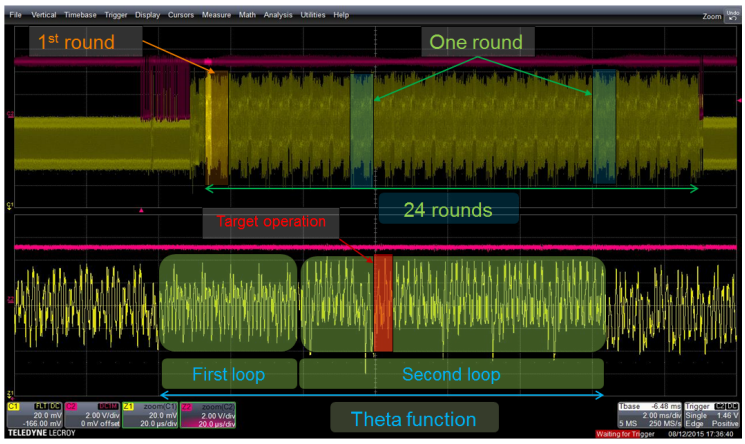

Fig. 5. Simple power analysis of the TUAK f_1 function.

When looking inside the first round, one can easily identify the θ function of the KECCAK permutation. Mainly, it is composed of two loops of 5 iterations each: the first loop consists in computing the temporary states and the second one stands for updating the state values with the XOR results. These observations are in-line with the source code of the KECCAK permutation detailed in Listing. 1 (*i.e.* the loops in Lines $(9-13)$ and Lines $(15-27)$).

For the sake of conciseness and simplicity, we restrict our practical side-channel analysis to target the third and the fourth 32-bit blocks of the subscriber key. We stress the fact that the same results were essentially obtained for the other key bytes and the TOP_c. As detailed in Sec. 3.1, the third and the fourth 32-bit blocks of the key are processed during the beginning of the second iteration of the second loop. This target area is highlighted in red at the bottom of Fig. 5.

3.3 Experimental Results

Once the target operation has been identified, we have first acquired 100.000 power consumption traces of 70.000 samples each. Then, the traces were synchronized by applying a cross-correlation technique [20] to remove the misalignment caused by inaccuracies in triggering the power measurements. In fact, we choose a unique signal pattern representing the beginning of the second loop of

[5] The pink curves describe the input/output signal.

the θ function (*e.g.* that detected on the first acquired trace). Then, for each trace, we search this pattern by computing the cross-correlation and then by taking the sample index that maximizes it. No further pre-processing techniques were applied since the noise level has been relatively low as one can see in Fig. 5 and the executed operations are easily identified. For illustration, we provide in Fig. 6 a superimposition of five synchronized traces.

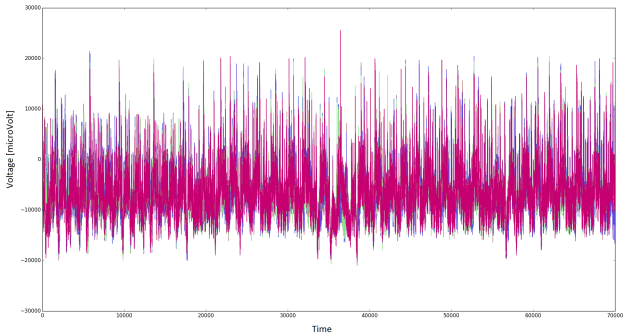

Fig. 6. Superimposition of five synchronized traces.

From Fig. 6, one can identify the last four iterations of the first loop of θ function followed by three iterations of the second loop. As described in Sec. 3.1, we will target the second iteration of the second loop which starts around the time sample 39.000 in Fig. 6.

Once the traces are aligned, we performed a first-order CPA attack using the 100.000 acquired traces. The attack results on the first key byte are shown in Fig. 7. The correlation peak was clearly sufficient to recover the good key byte value without ambiguity. Moreover, the correlation peak appears around the time sample 40.000 and this is in-line with our previous observations concerning the targeted attack area. The same results were obtained for the other targeted key bytes. For the sake of clarity, these results are not shown.

Finally, to estimate the minimum number of traces needed to disclose the key byte values through a side-channel analysis, we conducted 50 independent CPA attacks by using 50 independent sets of 2.000 power traces each. The evolution of the averaged ranks of the correct target key bytes is plotted in Fig. 8.

From Fig. 8, one can conclude that 700 traces are roughly needed to disclose the eight correct key byte values. The same results were obtained when targeting the XOR operations involving the TOP_c bytes. In addition and similarly to [24], we have estimated the complexity and the time needed to perform our attack including: the power traces acquisition, the traces alignment and the CPA attack processing. In fact, about 2 hours are roughly sufficient to recover the secret

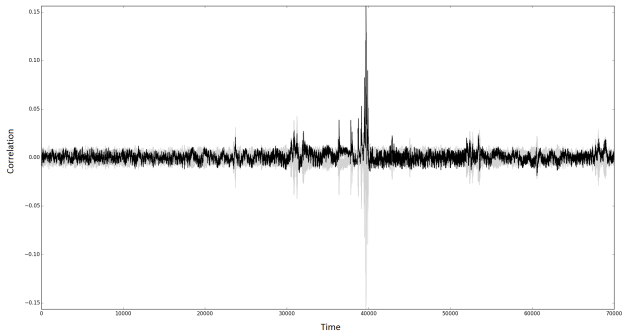

Fig. 7. CPA attack results when targeting the first key byte.

fig/GE-eps-converted-to.pdf

Fig. 8. Evolution of the correct key rank according to the number of observations.

values used during the authentication protocol and hence to procure a cloned USIM card. Relatively cheap to perform, our attack strategy raises a warning

flag to alert embedded systems developers to give further considerations when implementing cryptographic functions inside a USIM card.

4 Countermeasures and Discussions

Our side-channel analysis have pinpointed the requirement of protecting the TUAK algorithm against classical side-channel attacks. In fact, embedded systems developers have access to a large panel of countermeasures and they should take advantage of the broad literature on secure cryptographic implementations. Amongst side-channel countermeasures one should consider to ensure protection is masking [6,14,19,27]. Indeed, a solution could consist in masking the TUAK f_1 function input states. Doing so, a first-order side-channel attack could be thwarted. Another way to counter SCA, is to use shuffling techniques [33] which consist in randomizing the execution order of independent operations. Even better, shuffling could be combined with masking to reinforce the protection [29]. Recently, authors in [9] have emphasized the use of secret sharing techniques to protect the KECCAK permutation function. To summarize, it is up to the designers to choose the suitable type of countermeasure that guarantees the best performance-security trade-off.

Another protection approach will consist, first, in setting the subscriber key at 256-bit, which leads to hardening our attack process. For instance, when the adversary tries to recover the first 32-bit block of TOP_c (*i.e.* $s[0]$) as described in Sec. 3.1, actually she will recover $s[0] \oplus s[20]$ where $s[20]$ is the sixth 32-bit block of key. Said differently, the attacker will recover a XOR of both secrets and hence he needs to perform a brute force attack to carry-out one secret and then to get the other one. Second, the f_5 function could systematically be executed despite the associated extra overhead. Doing so, attacking the first and the second 32-bit blocks of the subscriber key become more challenging since the SQN is protected by the anonymity key (AK) as described in Sec. 2.

We believe that the latter defensive strategy is insufficient to completely counter our attack; it only aims at making it more complicated. In fact, albeit increasing the key size and executing the f_5 function, the TUAK implementation still leaks information about its secret values. For instance, recovering the third and the fourth 32-bit blocks of the key ($s[18]$ and $s[19]$) is still possible with our proposal: the adversary can apply the same procedure described in Sec. 3.1. Therefore, we emphasize combining the latter protection approach and some side-channel countermeasures to guarantee an acceptable security level.

5 Conclusion

In this work, we have demonstrated that with a classical CPA attack it is possible to break an unprotected software implementation of the TUAK algorithm being part of the authentication protocol of the 3G/4G mobile cellular networks. In fact, we have shown that the attack complexity is quite conservative and few

hours are needed to recover the secret data if the state-of-the-art countermeasures are not considered. Through this side-channel analysis, we want to put more emphasis on taking advantage of the existing research investigations on dedicated protections against side-channel attacks. Indeed, the security designers are free to choose the most adequate countermeasure with respect to the expected performance requirements for the authentication protocol. What matters is to follow the 3GPP recommendations and to take into account the physical security issues when implementing cryptographic algorithms in USIM cards.

References

1. ETSI TS 133 105; universal mobile telelecommunications system (UMTS); LTE; 3G security; cryptographic algorithm requirements (3GPP TS 33.105 version 13.0.0 release 13), 01/2016.
2. ETSI TS 133 202; universal mobile telelecommunications system (UMTS); LTE; 3G security; specification of the 3GPP confidentiality and integrety algorithms; document 2: Kasumi specification (3GPP TS 35.202 version 13.0.0 release 13), 01/2016.
3. ETSI TS 135 201; universal mobile telelecommunications system (UMTS); LTE; 3G security; specification of the 3GPP confidentiality and integrety algorithms; document 1: f_8 and f_9 specification (3GPP TS 35.201 version 13.0.0 release 13), 01/2016.
4. 3GPP specification: 135.206 (specification of the MILENAGE algorithm set), V13.0.0, 01/2016.
5. 3GPP specification: 135.231 (specification of the TUAK algorithm set), V13.0.0, 01/2016.
6. M.-L. Akkar and C. Giraud. An Implementation of DES and AES Secure against Some Attacks. In LNCS, editor, *Proceedings of CHES'01*, volume 2162 of *LNCS*, pages 309–318. Springer, May 2001. Paris, France.
7. S. Alt, P.-A. Fouque, G. Macario-rat, C. Onete, and B. Richard. A cryptographic analysis of UMTS/LTE AKA. Cryptology ePrint Archive, Report 2016/371, 2016. To appear in the proceedings of ACNS 2016.
8. E. Barkan, E. Biham, and N. Keller. *Instant Ciphertext-Only Cryptanalysis of* GSM *Encrypted Communication*, pages 600–616. Springer, 2003.
9. G. Bertoni, J. Daemen, N. Debande, T. Le, M. Peeters, and G. V. Assche. Power analysis of hardware implementations protected with secret sharing. In *45th Annual IEEE/ACM, MICRO 2012, Workshops Proceedings, Vancouver, BC, Canada, December 1-5, 2012*, pages 9–16, 2012.
10. G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche. Note on side-channel attacks and their countermeasures. Comment on the NIST Hash Competition Forum, May 2009.
11. G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche. The KECCAK reference, January 2011.
12. G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche. KECCAK implementation overview, Version 3.2, May 29, 2012.
13. É. Brier, C. Clavier, and F. Olivier. Correlation Power Analysis with a Leakage Model. In *CHES*, volume 3156 of *LNCS*, pages 16–29. Springer, August 11–13 2004. Cambridge, MA, USA.

14. S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In *CRYPTO*, volume 1666 of *LNCS*. Springer, August 15-19 1999. Santa Barbara, CA, USA. ISBN: 3-540-66347-9.
15. S. Chari, J. R. Rao, and P. Rohatgi. Template Attacks. In *CHES*, volume 2523 of *LNCS*, pages 13–28. Springer, August 2002. San Francisco Bay, USA.
16. J. Daemen and V. Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, 2002.
17. J. Doget, E. Prouff, M. Rivain, and F.-X. Standaert. Univariate side channel attacks and leakage modeling. *J. Cryptographic Engineering*, 1(2):123–144, 2011.
18. B. Gierlichs, L. Batina, P. Tuyls, and B. Preneel. Mutual information analysis. In *CHES, 10th International Workshop*, volume 5154 of *Lecture Notes in Computer Science*, pages 426–442. Springer, August 10-13 2008. Washington, D.C., USA.
19. L. Goubin and J. Patarin. DES and Differential Power Analysis. The "Duplication" Method. In *CHES*, LNCS, pages 158–172. Springer, Aug 1999. Worcester, USA.
20. N. Homma, S. Nagashima, T. Sugawara, T. Aoki, and A. Satoh. A High-Resolution Phase-Based Waveform Matching and Its Application to Side-Channel Attacks. *IEICE Transactions*, 91-A(1):193–202, 2008. New Orleans, Louisiana, USA. DOI: 10.1109/ISCAS.2007.378024.
21. P. C. Kocher, J. Jaffe, and B. Jun. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *Proceedings of CRYPTO'96*, volume 1109 of *LNCS*, pages 104–113. Springer-Verlag, 1996.
22. P. C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In *CRYPTO*, volume 1666 of *LNCS*, pages pp 388–397. Springer, 1999.
23. J. Liu, Y. Yu, F.-X. Standaert, Z. Guo, D. Gu, W. Sun, Y. Ge, and X. Xie. Cloning 3g/4g sim cards with a pc and an oscilloscope: Lessons learned in physical security. In *BlackHat*, 2015.
24. J. Liu, Y. Yu, F.-X. Standaert, Z. Guo, D. Gu, W. Sun, Y. Ge, and X. Xie. Small tweaks do not help: Differential power analysis of milenage implementations in 3g/4g usim cards. In G. Pernul, P. Y. A. Ryan, and E. R. Weippl, editors, *ESORICS (1)*, volume 9326 of *Lecture Notes in Computer Science*, pages 468–480. Springer, 2015.
25. P. Luo, Y. Fei, X. Fang, A. A. Ding, D. R. Kaeli, and M. Leeser. Side-channel analysis of mac-keccak hardware implementations. In *Proceedings of the Fourth HASP*, pages 1:1–1:8, New York, NY, USA, 2015. ACM.
26. K. Mayes, S. Babbage, and A. Maximov. Performance evaluation of the new TUAK mobile authentication algorithm. In *The Eleventh International Conference on Systems ICONS 2016*, pages 38–44, 2016. Related to work done in support of the ETSI SAGE group for mobile authentication standards.
27. T. S. Messerges. Securing the AES Finalists Against Power Analysis Attacks. In *FSE'00*, pages 150–164. Springer-Verlag, April 2000. New York.
28. J. R. Rao, P. Rohatgi, H. Scherzer, and S. Tinguely. Partitioning attacks: Or how to rapidly clone some GSM cards. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, SP '02, pages 31–, Washington, DC, USA, 2002.
29. M. Rivain, E. Prouff, and J. Doget. Higher-Order Masking and Shuffling for Software Implementations of Block Ciphers. In *CHES*, volume 5747 of *LNCS*, pages 171–188. Springer, September 6-9 2009. Lausanne, Switzerland.
30. W. Schindler. Advanced stochastic methods in side channel analysis on block ciphers in the presence of masking. *Journal of Mathematical Cryptology*, 2(3):291–310, October 2008.

31. W. Schindler, K. Lemke, and C. Paar. A Stochastic Model for Differential Side Channel Cryptanalysis. In LNCS, editor, *CHES*, volume 3659 of *LNCS*, pages 30–46. Springer, Sept 2005. Edinburgh, Scotland, UK.
32. M. M. I. Taha and P. Schaumont. Side-channel analysis of mac-keccak. In *2013 IEEE International Symposium on Hardware-Oriented Security and Trust, HOST 2013, Austin, TX, USA, June 2-3, 2013*, pages 125–130, 2013.
33. N. Veyrat-Charvillon, M. Medwed, S. Kerckhof, and F.-X. Standaert. Shuffling against Side-Channel Attacks: A Comprehensive Study with Cautionary Note. In X. Wang and K. Sako, editors, *ASIACRYPT*, volume 7658 of *Lecture Notes in Computer Science*, pages 740–757. Springer, 2012.
34. D. Wagner, B. Schneier, and J. Kelsey. *Cryptanalysis of the cellular message encryption algorithm*, pages 526–537. Springer, 1997.
35. M. Zohner, M. Kasper, M. Stottinger, and S. Huss. Side channel analysis of the sha-3 finalists. In *DATE 2012*, pages 1012–1017, March 2012.