How Deep Learning Helps Compromising USIM*

Martin Brisfors, Sebastian Forsmark, and Elena Dubrova

Royal Institute of Technology (KTH), Stockholm, Sweden
{brisfors,sforsm,dubrova}@kth.se

Abstract. It is known that secret keys can be extracted from some USIM cards using Correlation Power Analysis (CPA). In this paper, we demonstrate a more advanced attack on USIMs, based on deep learning. We show that a Convolutional Neural Network (CNN) trained on one USIM can recover the key from another USIM using at most 20 traces (four traces on average). Previous CPA attacks on USIM cards required high-quality oscilloscopes for power trace acquisition, an order of magnitude more traces from the victim card, and expert-level skills from the attacker. Now the attack can be mounted with a $1000 budget and basic skills in side-channel analysis.

Keywords: USIM · MILENAGE · AES · Power analysis · Deep learning.

1 Introduction

Today the Universal Subscriber Identity Module (USIM) card is perceived as an enabler for security, privacy, and trust in services and applications provided by the mobile communication networks. The USIM card is the only platform which is used to secure network access in Universal Mobile Telecommunications Service (UMTS) and Long-Term Evolution (LTE) cellular networks and it remains an essential part of New Radio (NG) cellular networks [23]. If the USIM's secret key is compromised, the attacker can decrypt the subscriber's communication, impersonate the subscriber, or impersonate the network.

SIM cards were introduced in 2G GSM systems in 1996 to address shortcomings discovered in previous analog systems and to meet emerging threats [14]. In particular, risk of fraud (making calls charged to other subscribers) was considered a major problem. The SIM enabled a subscriber's authentication using a pre-shared secret key stored in the SIM and, consequently, a correct charging.

However, the algorithm COMP-128, which implements a subscriber's authentication (A3) and session key generation (A8) functions defined by GSM standard [15], was soon broken. Originally confidential, the first version of COMP-128 was published in 1998 [18] and the second and the third in 2013 [24]. All three versions are based on a compression function which has a weakness in diffusion of the second round. This allows for a "narrow pipe" attack which can recover

* Supported by the research grant No 2018-04482 from the Swedish Research Council.

the secret key from the SIM from 131K challenge-response pairs [13]. An improved attack which can recover the key from 20K challenge-response pairs was presented at DEFCON 2004 [16].

To improve security, USIM cards were launched in 3G UMTS systems in 2001. In particular, procedures for mutual authentication of the subscriber and the core network based on the Authentication and Key Agreement (AKA) protocol were introduced. The mutual authentication was intended to mitigate threats of rogue radio base stations.

The main structure of a USIM-based access authentication remains intact in 4G and 5G. The 4G LTE standard uses the same mutual authentication scheme as UMTS. The latest 5G NG standard uses a hardened version, called 5G-AKA, as one of its mandatory authentication options [1, 21].

The 3rd Generation Partnership Project (3GPP) recommends implementing the authentication and key generation functions of AKA using either MILENAGE based on AES [2], or TUAK based on SHA3 [3]. In this paper we focus on attacks on MILENAGE algorithm.

Previous work. Implementations of MILENAGE in some commercial USIM cards are known to be vulnerable to Correlation Power Analysis (CPA) [10, 17]. In [10], eight commercial USIM cards were analyzed. It was shown that it is possible to recover the secret key as well as other secret parameters required to clone the USIM card using 200 to 1000 power traces from the target USIM, depending on USIM's type. In [10], 9 commercial USIM cards were analyzed and one was found vulnerable. The secret key was recovered using 4000 power traces.

Our contribution. We demonstrate the first attack on the implementation of MILENAGE in a commercial USIM card based on deep learning-based power analysis. The deep learning-based power analysis has three important advantages over CPA:

1. It requires an order of magnitude fewer traces from a victim USIM card.
2. It can bypass some countermeasures, e.g. jitter [7] and masking [12], and thus break some USIMs which cannot be broken by CPA.
3. It does not require expert-level skills in side-channel analysis (if a trained neural network and all scripts for the attack stage are available).

In addition, we believe that we are the first to break a commercial USIM using equipment with the total cost below $1000 (excluding PC). In a combination with (3), the latter makes the presented attack particularly threatening.

2 USIM Security

In this section, we describe key terms related to cellular networks and USIM cards, AKA protocol and MILENAGE algorithm.

We use the terms 2G, 3G, 4G and 5G to refer to the corresponding generations of the 3GPP defined systems. We typically use the terminology of 4G.

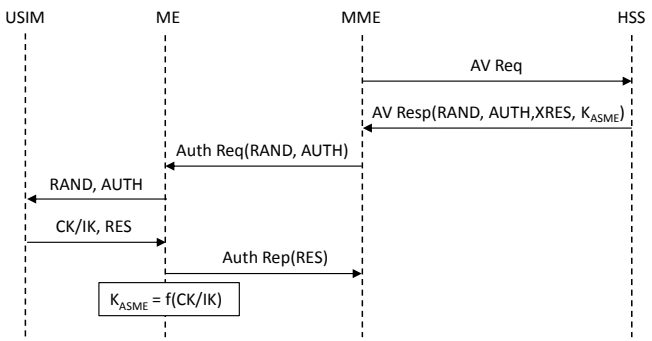


Fig. 1. The AKA protocol.

2.1 Network Structure

A mobile network consists of the following three essential types of entities:

- *Mobile Entity* (ME) equipped with a USIM.
- An authenticator in the serving network, called the *Mobility Management Entity* (MME).
- A database in the home network storing the subscriber credentials, called the *Home Subscriber Server* (HSS).

The USIM contains all necessary information about the subscription at the home network, including an *International Mobile Subscriber Identity* (IMSI). The USIM also stores a long-term key K which is pre-shared with the HSS. All cryptographic operations which involve K are carried out within the USIM.

The home network and the serving network are usually connected over a secure channel, such as IPsec or Transport Layer Security (TLS).

2.2 Authentication and Key Agreement Protocol

The 3GPP AKA is a challenge-response multi-party protocol based on symmetric key cryptography. The main steps are (see Fig. 1) [5]:

1. The MME initiates AKA by sending a request for an *Authentication Vector* (AV) associated with a particular subscriber in the HSS.
2. The HSS responds with an AV consisting of the 4-tuple (RAND, AUTN, XRES, K_{ASME}), where RAND is a random value, AUTN is a network authentication token, XRES is the expected response from the ME and K_{ASME} is the session key which is established in the ME and MME when the AKA run is completed. The AUTN, XRES and K_{ASME} are derived from the RAND and K. As an intermediate step in the derivation of K_{ASME}, the cipher key CK and integrity key IK are produced.
3. When the MME gets the AV, it initiates the authentication procedure with the ME by forwarding the RAND and AUTN.

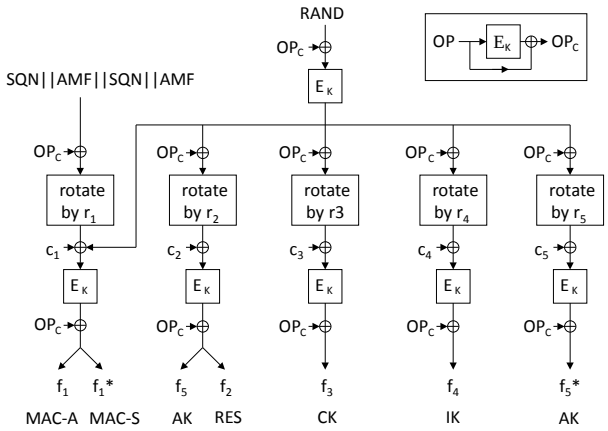


Fig. 2. The MILENAGE algorithm.

4. The ME sends the RAND and AUTN to the USIM, which verifies the authenticity and freshness of the AUTN parameter. If the verification succeeds, the USIM derives a response parameter RES and keys CK and IK from K, RAND and AUTN.
5. The USIM forwards the RES, CK and IK to the ME.
6. The ME derives K_{ASME} from the CK and IK and sends RES to the MME.
7. The MME verifies that RES is equal to XRES. If the verification is successful, MME accepts the authentication. Otherwise it rejects the authentication.

AKA uses 48-bit sequence numbers to prove the freshness of the RAND and AUTN parameters to the ME and USIM. The sequence number SQN is included in the AUTN parameter. If the sequence number is old, the ME or USIM rejects the authentication. AKA also includes a re-synchronization mechanism for sequence numbers.

The AKA protocol described above is a 4G AKA. Mutual authentication scheme of 3G AKA is the same. The 3G AKA and 4G AKA differ in the key agreement part. In 3G, the keys CK and IK are used to protect traffic, while in LTE they are used to derive a tree of keys. 2G systems can also use 3G AKA.

In 5G-AKA, the response RES computed in step 4 is processed one more time through a key derivation function (HMAC-SHA256) depending on RES, RAND, and CK || IK. The rest of the mutual authentication scheme is similar.

2.3 MILENAGE Algorithm

The 3GPP MILENAGE algorithm [2, 4], implements seven security functions related to authentication and key agreement: $f_1, f_1^*, f_2, f_3, f_4, f_5, f_5^*$. The block diagram of the MILENAGE is shown in Fig. 2.

The functions f_1 and f_1^* compute a 64-bit, 128-bit or 256-bit network authentication code MAC-A and resynchronisation authentication code MAC-S,

respectively. The function f_2 generates a 128-bit RES. The functions f_3 computes a 128-bit or 256-bit cipher key CK and integrity key IK, respectively. The functions f_5 and f_5^* generate a 48-bit anonymity key AK. AK is used to derive the sequence number SK.

The block denoted by E_K represents a 128-bit block cipher keyed with 128-bit or 256-bit key K. 3GPP recommends AES for implementing the block cipher.

The OP is an *Operator Variant Algorithm Configuration Field* defined by an operator and used for all subscribers. The OP_C is computed from OP and K as shown in the box at the top right corner of Fig. 2. There two alternative options for computing OP_C on the USIM which, in turn, determine where OP is stored:

- OP_C is computed off the USIM during the USIM pre-personalization process and stored on the USIM. In this case OP is not stored on the USIM.
- OP_C is computed on the USIM each time it is required in the algorithm. In this case OP is stored on the USIM.

A 128-bit random challenge RAND and a 16-bit Authentication Management Field AMF are controlled by the home network. A 48-bit sequence numbers SQN is a shared counter which is incremented with each successful authentication, to protect against replay attacks.

The parameters r_1, r_2, r_3, r_4, r_5 are integers in the range 0 - 127 which defines amounts by which intermediate variables are cyclically rotated. The parameters c_1, c_2, c_3, c_4, c_5 are 128-bit constants which are XORed with intermediate variables. Default values for $r_1 - r_5$ and $c_1 - c_5$ are set in the specification, however an operator may select different values for these constants to customize the algorithm further.

3 Attacking MILENAGE

3.1 Measurement Setup

Our measurement setup is shown in Fig. 3. It consists of the ChipWhisperer-Lite board, the CW308 UFO board and a custom smartcard reader for the ChipWhisperer called LEIA.

The ChipWhisperer is a hardware security evaluation toolkit based on a low-cost open hardware platform and an open source software [19]. It can be used to measure power consumption with a maximum sampling rate of 105 MS/sec.

The CW308 UFO board is a generic platform for evaluating multiple targets [9]. The target board is plugged into a dedicated U connector.

LEIA is the ISO7816 interface smart card reader compatible with the ChipWhisperer [11]. The reader is implemented on a STM32 microcontroller. It controls the clock and I/O lines to the smartcard under test and provides a trigger output to command the ChipWhisperer to start an acquisition. The reader has a resistor placed between the load and ground to enable voltage measuring.

The open source software [8] is run on a PC to control the hardware and execute the MILENAGE algorithm. The PC plays the role of MME. To initiate the authentication, the PC communicates with the USIM using the commands in Application Protocol Data Unit (APDU) language.

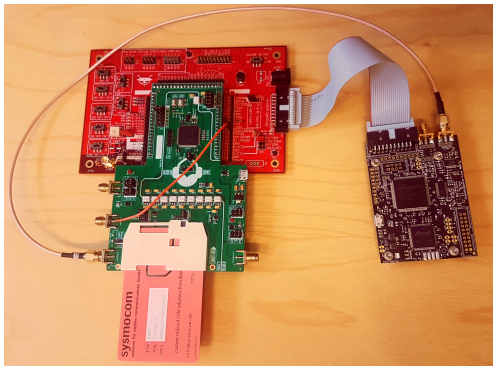


Fig. 3. Equipment for trace acquisition.

3.2 Attack Target

We used the commercial Sysmocom USIM cards sysmoSIM-SJS1 [25] as an attack target. These cards are compliant with 2G GSM, 3G UMTS and 4G LTE standards and can be used with any core network. The 2G SIM authentication algorithm is set to COMP128v1 and the 3G/UMTS authentication algorithm to MILENAGE [25]. All cards ship with a factory-default, card-individual random K and $\mathrm{OP_C}$. However, the cards are fully re-programmable and the user can re-program K, OP or $\mathrm{OP_C}$.

Sysmocom USIM cards were one of the target cards in the CPA attack presented in [10]. The CPA attack presented in [10] used 4000 power traces to recover the key from a Sysmocom USIM card.

3.3 Trace Acquisition

In order to find a time interval containing the attack points, we used a high-end LeCroy oscilloscope with the maximum sampling rate 10 GS/sec and 1 GHz bandwidth per channel. If only low-cost equipment such as ChipWhisperer-Lite is available, the same result can be achieved by capturing at a lower sampling rate and shifting the offset until the distinct shape of AES encryption is found.

To capture a single trace, the following three steps are repeated in the script for trace acquisition:

1. Select a random RAND 16-byte value.
2. Call MILENAGE algorithm with the RAND as seed. This turns on a trigger signal which stays high until a response is received.
3. Capture and save the power trace as well as RAND.

Fig. 4 shows a full power trace of USIM card for one call of MILENAGE. Fig. 5 presents a zoomed interval of the trace in which seven AES encryptions can be clearly distinguished. Note that MILENAGE calls the encryption E_k only six times (see Fig. 2). However E_k is also used to compute $\mathrm{OP_C}$ (see the box at

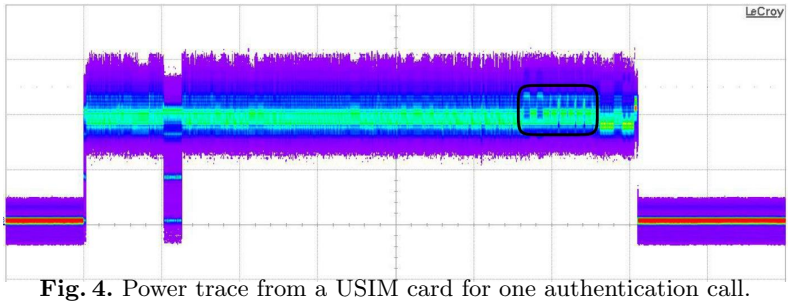


Fig. 4. Power trace from a USIM card for one authentication call.

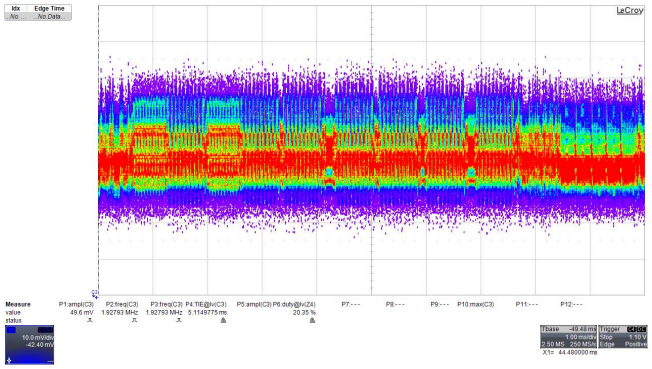


Fig. 5. Zoomed interval of the trace in Fig.4 representing the MILENAGE execution.

the top right corner of Fig. 2). Even if OP_C is programmed into the USIM, it might by possible that the USIM performs the OP_C computation step anyway and then discards the result.

Since is was not clear whether the MILENAGE algorithm starts at the first or at the second AES encryption, we applied CPA to both time intervals. The CPA successfully recovered the key from 300 traces corresponding to the second AES encryption and failed on the first AES encryption. From this we concluded that the MILENAGE algorithm starts at the second AES encryption[1].

ChipWhisperer is well-known for its ability to perfectly synchronize traces. However, traces captured by ChipWhisperer in our experiments were desynchronized. Since ChipWhisperer controls LEIA and LEIA controls the USIM, when ChipWhisperer sends a command to the USIM to start MILENAGE, the command has to be forwarded by LEIA. The forwarding may cause desynchronization.

Previous work on deep learning side-channel attacks on AES [22] has shown that desynchronized traces may be used directly if a CNN model is used for the attack. It is also possible to synchronize the traces before training and testing, regardless of the model's type. In our experiments we tried both approaches.

[1] In previous CPA attack on Sysmocom USIM cards [10] it was suggested that MILENAGE starts at the first AES encryption, but our results indicate otherwise.

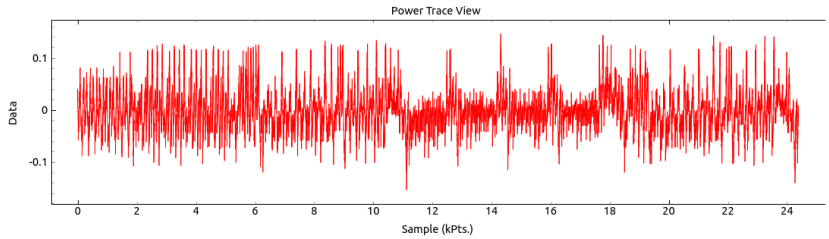


Fig. 6. Power trace representing the first round of AES (within 6KPt - 19KPt).

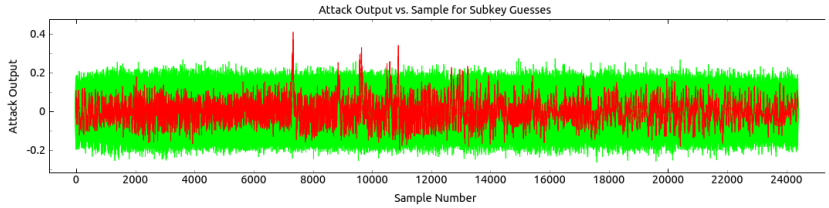


Fig. 7. Correlation results for the 10th byte of $\mathrm{OP_C} \oplus \mathrm{K}$.

3.4 Recovering Key from Profiling USIM by CPA

To train a deep learning model, we need a profiling USIM with known parameters K, $\mathrm{OP_C}$, $r_1 - r_5$ and $c_1 - c_5$, The target USIMs in our experiments use default values for the constants $r_1 - r_5$ and $c_1 - c_5$[2]. They are also re-programmable, so we could simply re-write factory-default K and $\mathrm{OP_C}$ by new values. However, our focus is on extracting the factory-default K and $\mathrm{OP_C}$ in order to estimate the number of traces required for a successful CPA attack on this type of card. Below we describe our CPA attack based on the Hamming weight power model.

We can see from Fig. 2 that, at the first step of MILENAGE, $\mathrm{RAND} \oplus \mathrm{OP_C}$ is computed and then the result is encrypted. If AES-128 is used for implementing the encryption $\mathrm{E_K}$, the 128-bit key K can be recovered as follows:

1. Use a USIM to execute MILENAGE for a large number of random challenges $\mathrm{RAND}_0, \ldots, \mathrm{RAND}_{n-1}$ and capture the resulting power traces $\boldsymbol{T} = \{T_0, \ldots, T_{n-1}\}$ as described in Section 3.3.
2. First, recover $\mathrm{OP_C} \oplus$ K by a CPA with the S-box output in the first round of AES as the attack point.
3. Second, recover the first round key, RK_1 by a CPA with the S-box output in the second round of AES as the attack point.
4. Using the key expansion algorithm of AES-128, deduce K from RK_1.
5. Compute $\mathrm{OP_C}$ as $\mathrm{OP_C} = (\mathrm{OP_C} \oplus \mathrm{K}) \oplus \mathrm{K}$.

[2] If a USIM uses non-default values for $r_1 - r_5$ and $c_1 - c_5$, they can be derived using the technique presented in [17].

A similar two-step strategy for recovering K and $\mathrm{OP_C}$ was used in the CPA attacks on MILENAGE presented in [10, 17]. Next we describe the process of recovering $\mathrm{OP_C}\oplus$ K in more details.

Fig. 6 shows a power trace representing the execution of the first round of AES. The four operations SubBytes(), ShiftRows(), MixColumns() and AddRoundKey() are executed approximately between the points 6KPt and 19KPt.

Let $\mathrm{RAND}_{j,k}$ denote the kth byte of RAND_j, $k \in \{0, \ldots, 15\}$, $j \in \{0, \ldots, n-1\}$. For each RAND_j and each possible value $v \in \{0, \ldots, 255\}$ of the kth byte of $\mathrm{OP_C}\oplus$ K, the power estimate $x_{j,v}$ for the trace $T_j \in \boldsymbol{T}$ and the guess v is calculated as the Hamming weight of the S-box output in the first round:

$$x_{j,v} = \mathrm{HW}(\text{S-box}[\mathrm{RAND}_{j,k} \oplus v]).$$

Let $y_{j,i}$ denote the data point i in the trace $T_j \in \boldsymbol{T}$, for $j \in \{0, \ldots, n-1\}$, $i \in \{0, \ldots, m-1\}$, where m is the number of data points in the trace. For the trace in Fig. 6, $m = 24.4$K (buffer size of ChipWhisperer-Lite).

To check how well the HW model and measured traces correlate for each guess v and data point i, Pearson correlation coefficients, $r_{v,i}$, for the data sets $\{x_{1,v}, ..., x_{n,v}\}$ and $\{y_{1,i}, ..., y_{n,i}\}$ are computed:

$$r_{v,i} = \frac{\sum_{j=0}^{n-1}(x_{j,v} - \bar{x}_v)(y_{j,i} - \bar{y}_i)}{\sqrt{\sum_{j=0}^{n-1}(x_{j,v} - \bar{x}_v)^2 \sum_{j=0}^{n-1}(y_{j,i} - \bar{y}_i)^2}},$$

where n is the sample size, and $\bar{x}_v = \frac{1}{n}\sum_{j=0}^{n-1} x_{j,v}$ and $\bar{y}_i = \frac{1}{n}\sum_{j=0}^{n-1} y_{j,i}$ are the sample means.

In our experiments, the maximum correlation coefficients for different bytes of $\mathrm{OP_C}\oplus$ K ranged in the interval 0.2758 - 0.4208. Fig. 7 shows correlation results for the 10th byte of $\mathrm{OP_C}\oplus$ K. One can clearly see distinct peaks.

Once all bytes of $\mathrm{OP_C}\oplus$ K are recovered, we can estimate the number of traces required for a successful attack using the Partial Guessing Entropy (PGE) [20]. When the PGEs of all bytes reach zero, the $\mathrm{OP_C}\oplus$ K is recovered. From Fig. 8, we can see that about 300 traces are required to recover the key in this attack.

The process of recovering $\mathrm{RK_1}$ by a CPA with the S-box output in the second round AES as an attack point is similar. The power estimate $x_{j,v}$ for the trace $T_j \in \boldsymbol{T}$, for $j \in \{0, \ldots, n-1\}$, and each $\mathrm{RK_1}$ subkey guess $v \in \{0, \ldots, 255\}$ is calculated as the Hamming weight of the S-box output in the second round for the input $state_{j,k} \oplus v$, where $state_{j,k}$ is the kth byte of the AES state $state_j$ after ShiftRows() and MixColumns() in the first round for the challenge RAND_j. Note that $\mathrm{RK_1}$ can only be recovered after all bytes of $\mathrm{OP_C}\oplus$ K are recovered because only then can ShiftRows() and MixColumns() be fully evaluated. This is necessary for determining the input to the second round.

3.5 Profiling Stage

At the profiling stage, the models $\mathrm{CNN}_{1,k}$ and $\mathrm{CNN}_{2,k}$ for recovering the bytes of $\mathrm{OP_C}\oplus$ K and $\mathrm{RK_1}$, respectively, are trained for all $k \in \{0, \ldots, 15\}$ as follows:

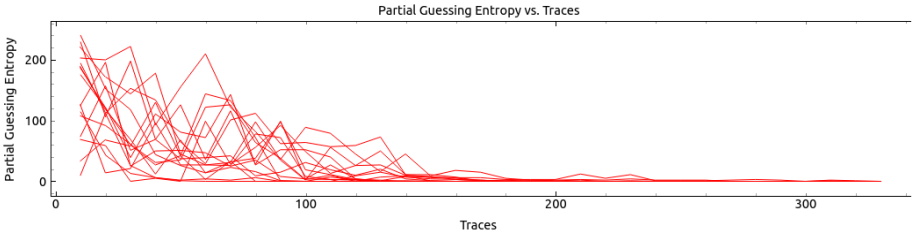


Fig. 8. Partial guessing entropy vs. number of traces.

1. Identify the start of MILENAGE execution in the trace of the profiling USIM (see Fig. 4 and 5). Set the offset for capture so that traces include the computation of S-box in the first round of AES.
2. Use the profiling USIM to execute MILENAGE for a large number n_p of random challenges $\text{RAND}_0, \ldots, \text{RAND}_{n_p-1}$ and capture the resulting set of traces $\boldsymbol{T}_{p,1} = \{T_0, \ldots, T_{n_p-1}\}$. Let m be the number of data points in each trace $T_j \in \boldsymbol{T}_{p,1}$.
3. For each $k \in \{0, \ldots, 15\}$:
   - Assign to each trace $T_j \in \boldsymbol{T}_{p,1}$ a label $l_k(T_j)$ equal to the value of the S-box output in the first round during the evaluation of the kth byte of $\text{RAND}_j \oplus (\text{OP}_\text{C} \oplus \text{K})$:

     $$l_k(T_j) = \text{S-box}[\text{RAND}_{j,k} \oplus (\text{OP}_\text{C} \oplus \text{K})_k],$$

     where $(\text{OP}_\text{C} \oplus \text{K})_k$ is the kth byte of $\text{OP}_\text{C} \oplus \text{K}$ and $\text{RAND}_{j,k}$ is the kth byte of the challenge RAND_j used to generate T_j. $\text{OP}_\text{C} \oplus \text{K}$ and $\text{RAND}_{j,k}$ is assumed to be known during profiling.
   - Use the labeled set of traces $\boldsymbol{T}_{p,1}$ to train a model $\text{CNN}_{1,k} : \mathbb{R}^m \to \mathbb{I}^{256}$, $\mathbb{I} := \{x \in \mathbb{R} \mid 0 \leq x \leq 1\}$, which takes as input a trace $T_j \in \mathbb{R}^m$ and produces as output a score vector $S_{j,k} = \text{CNN}_{1,k}(T_j) \in \mathbb{I}^{256}$ in which the value of the ith element, $s_{j,k,i}$, is the probability that the S-box output in the first round is equal to $i \in \{0, \ldots, 255\}$ when the kth byte of $\text{RAND}_j \oplus (\text{OP}_\text{C} \oplus \text{K})$ is processed:

     $$s_{j,k,i} = \Pr(\text{S-box}[\text{RAND}_{j,k} \oplus (\text{OP}_C \oplus \text{K})_k] = i).$$

     The training strategy is described in Section 4.1.
4. Once all bytes of $\text{OP}_\text{C} \oplus \text{K}$ are recovered, use the profiling USIM to capture a large set of power traces $\boldsymbol{T}_{p,2}$ including the interval corresponding to the execution of S-box in the second round of AES.
5. For each $k \in \{0, \ldots, 15\}$:
   - Assign to each trace $T_j \in \boldsymbol{T}_{p,2}$ a label $l_k(T_j)$ equal to the value of the S-box output in the second round during the evaluation of the kth byte of $state_j \oplus \text{RK}_1$:

     $$l_k(T_j) = \text{S-box}[state_{j,k} \oplus \text{RK}_{1,k}],$$

where $\mathrm{RK}_{1,k}$ is the kth subkey of the RK_1 and $state_{j,k}$ is the kth byte of $state_j$ after ShiftRows() and MixColumns() in the first round for the challenge RAND_j. The value of $state_k$ is known for any RAND_j once all bytes of $\mathrm{OP_C}\oplus\mathrm{K}$ are recovered.

– Use the labeled set of traces $\boldsymbol{T}_{p,2}$ to train a model $\mathrm{CNN}_{2,k} : \mathbb{R}^m \rightarrow \mathbb{I}^{256}$ which takes as input a trace $T_j \in \mathbb{R}^m$ and produces as output a score vector $S_{j,k} = \mathrm{CNN}_{2,k}(T_j) \in \mathbb{I}^{256}$ in which the value of the ith element, $s_{j,k,i}$, is the probability of that the S-box output in the second round is equal to $i \in \{0, \ldots, 255\}$ when the kth byte of $state_j \oplus \mathrm{RK}_1$ is processed:

$$s_{j,k,i} = \Pr(\text{S-box}[state_{j,k} \oplus \mathrm{RK}_{1,k}] = i).$$

The training strategy is described in Section 4.1.

3.6 Attack Stage

At the attack stage, the trained models $\mathrm{CNN}_{1,k}$ and $\mathrm{CNN}_{2,k}$ are used to recover the kth byte of $\mathrm{OP_C}\oplus$ K and RK_1, respectively, for all $k \in \{0, \ldots, 15\}$.

To recover the bytes of $\mathrm{OP_C}\oplus$ K:

1. Identify the start of MILENAGE execution in the trace of the victim USIM (see Fig. 4 and 5). Set the offset for capture so that traces include the computation of S-box in the first round of AES.
2. Use the victim USIM to execute MILENAGE for a small number n_a of random challenges $\mathrm{RAND}_0, \ldots, \mathrm{RAND}_{n_a-1}$ and capture the resulting set of traces $\boldsymbol{T}_{a,1} = \{T_0, \ldots, T_{n_a-1}\}$, with m data points in each trace where m is the input size of $\mathrm{CNN}_{1,k}$.
3. For each $k \in \{0, \ldots, 15\}$, use the model $\mathrm{CNN}_{1,k}$ to classify the traces of an ordered set $\boldsymbol{T}_{a,1}$. For each $j \in \{0, \ldots, n_a - 1\}$, the trace $T_j \in \boldsymbol{T}_{a,1}$ is classified as

$$\tilde{l} = \underset{i \in \{0,\ldots,255\}}{\arg\max} \left(\prod_{l=0}^{j} s_{l,k,i}\right),$$

where $s_{l,k,i}$ is the ith element of the score vector $S_{l,k} = \mathrm{CNN}_{1,k}(T_l)$ of a trace $T_l \in \boldsymbol{T}_{a,1}$ which precedes T_j in $\boldsymbol{T}_{a,1}$. Once $\tilde{l} = l_k(T_j)$, the classification is successful. The kth byte of $\mathrm{OP_C}\oplus$ K is then recovered as

$$(\mathrm{OP_C} \oplus \mathrm{K})_k = \text{S-box}^{-1}(l_k(T_j)) \oplus \mathrm{RAND}_{j,k}.$$

where $\mathrm{RAND}_{j,k}$ is the kth byte of RAND_j used to generate the trace T_j.

The subkeys of the round key RK_1 are recovered similarly using the models $\mathrm{CNN}_{2,k}$, $k \in \{0, \ldots, 15\}$, except that the offset for capture is selected so that traces of the set $\boldsymbol{T}_{a,2} = \{T_0, \ldots, T_{n_a-1}\}$ include the computation of S-box in the second round of AES. Once a trace $T_j \in \boldsymbol{T}_{a,2}$ is successfully classified as $\tilde{l} = l_k(T_j)$ for some $j \in \{0, \ldots, n_p - 1\}$, the kth byte of RK_1 is recovered as

$$\mathrm{RK}_{1,k} = \text{S-box}^{-1}(l_k(T_j)) \oplus state_{j,k},$$

where $state_{j,k}$ is the kth byte of $state_j$ after ShiftRows() and MixColumns() in the first round for the challenge RAND_j.

Finally, K is derived from RK_1 using the key expansion algorithm of AES.

4 Experimental Results

In the experiments, we used two identical USIMs of type described in Section 3.2. One card was used for profiling and another - for attack. In the section, we refer to these cards as USIM_p and USIM_a, respectively.

Using the equipment and the method described in Sections 3.1 and 3.3, we captured two large sets of traces of size $n_p = 35.5\text{K}$ from USIM_p - one for the first round, $\boldsymbol{T}_{p,1}$, and another the second round of AES, $\boldsymbol{T}_{p,2}$. Similarly, we captured two smaller sets of traces, $\boldsymbol{T}_{a,1}$ and $\boldsymbol{T}_{a,2}$, of size $n_a = 6\text{K}$ from USIM_a. Each trace in these four sets contains $m = 24.4\text{K}$ data points.

4.1 Training Process

Initially, we tried three strategies: (1) Training an MLP model on synchronized traces; (2) Training a CNN model on synchronized traces; (3) Training a CNN model on de-synchronized traces.

Early testing showed that both MLP and CNN models trained on traces from USIM_p can recover subkeys from USIM_a. It was also possible to recover the key using a CNN model trained on de-synchronized traces. However, the CNN model trained on synchronized traces has shown significantly better results than the other two, so we focused on it.

We used 70% of traces from USIM_p for training and 30% for validation. We searched for the best options for learning rate, number of epochs, and number of layers. A lot of different sizes of filter kernels were tried for the convolution layers, and different input sizes were tested.

After many experiments, we settled for the model with the architecture shown in Table 1. The model was trained for 100 epochs using batch size 100. The RMSprop optimizer with a learning rate of 0.0001 and no learning rate decay was used. No dropout was used. We found it is best to train on the entire buffer of ChipWhisperer, which contains 24.4K data points. For this reason, the input layer size of the model in Table 1 is 24.4K. The first kernel is quite large because the execution of S-box operation takes a large number of data points (approx. 900). The shape of a power trace suggests that a single S-box operation is performed on four bytes at a time, i.e. this USIM uses a 32-bit microcontroller.

To train the model $\text{CNN}_{1,k}$ which classifies the kth byte of $\text{OP}_\text{C} \oplus \text{K}$, each trace $T_j \in \boldsymbol{T}_{p,1}$ is assigned a label equal to the value of the S-box output for the input $\text{RAND}_{j,k} \oplus (\text{OP}_\text{C} \oplus \text{K})_k$, where $\text{RAND}_{j,k}$ is the kth byte of the challenge RAND_j used to generate T_j, for all $k \in \{0, \ldots, 15\}$ and $j \in \{0, \ldots, n_p - 1\}$.

To train the model $\text{CNN}_{2,k}$ which classifies the kth subkey of RK_1, each trace $T_j \in \boldsymbol{T}_{p,2}$ is assigned a label equal to the value of the S-box output for the input

Layer Type	Output Shape	Parameter #
Input (Dense)	(None, 24400, 1)	0
Conv1D 1	(None, 24400, 8)	7208
AveragePooling1 1	(None, 3050, 8)	0
Conv1D 2	(None, 3050, 16)	1296
AveragePooling1 2	(None, 305, 16)	0
Conv1D 3	(None, 305, 32)	5152
AveragePooling1 3	(None, 61, 32)	0
Conv1D 4	(None, 61, 64)	18496
AveragePooling1 4	(None, 6, 64)	0
Conv1D 5	(None, 4, 128)	24704
AveragePooling1 5	(None, 1, 128)	0
Flatten	(None, 128)	0
Dense 1	(None, 300)	38700
Output (Dense)	(None, 256)	77056

Total Parameters: 172,612

Table 1. Architecture of the best CNN model.

$state_{j,k} \oplus \mathrm{RK}_{1,k}$, where $state_{j,k}$ is the kth byte of $state_j$ after ShiftRows() and MixColumns() in the first round, for all $k \in \{0, \ldots, 15\}$ and $j \in \{0, \ldots, n_p - 1\}$.

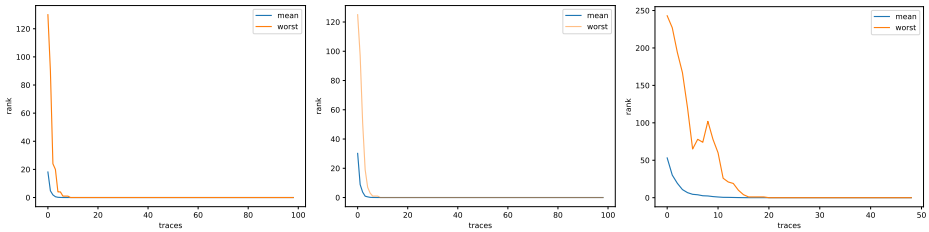
We would like to mention that it might possible to train a *single* neural network capable of recovering any byte of $\mathrm{OP_C} \oplus \mathrm{K}$ (or RK_1). Such a possibility has been already demonstrated for an 8-bit microcontroller implementation of AES-128 [6]. The MLP model presented in [6] can recover all subkeys from the target device different from the profiling device. The MLP was trained on a union of 16 sets of power traces $\boldsymbol{T}_k = \{T_{k,1}, \ldots, T_{k,n}\}$, $k \in \{0, \ldots, 15\}$, such that the trace $T_{k,j} \in \boldsymbol{T}_k$, $j \in \{0, \ldots, n-1\}$, contains data points in which S-box evaluates the kth byte of $\mathrm{P} \oplus \mathrm{K}$, where P is the plaintext. However, taking into account that our USIM implementation of AES-128 is based on a 32-bit microcontroller, we have chosen to train the networks separately for each byte position.

4.2 Testing Results

For any $k \in \{0, 1, \ldots, 15\}$, the $\mathrm{CNN}_{1,k}$ with the architecture shown in Table 1 trained on traces $\boldsymbol{T}_{p,1}$ from USIM_p successfully recovers the kth byte of $\mathrm{OP_C} \oplus \mathrm{K}$ from at most 10 traces $\boldsymbol{T}_{a,1}$ of USIM_a. The byte number does not seem to matter. As an example, Fig. 9(a) and (b) show the worst and the average ranks of models $\mathrm{CNN}_{1,0}$ and $\mathrm{CNN}_{1,5}$. One can see that the plots are similar. For $k = 0$, the average number of traces is 2.24. For $k = 5$, it is 2.5.

Similarly, for any $k \in \{0, 1, \ldots, 15\}$, the $\mathrm{CNN}_{2,k}$ with the architecture shown in Table 1 trained on traces $\boldsymbol{T}_{p,2}$ from USIM_p successfully recovers the kth subkey of RK_1 from at most 20 traces $\boldsymbol{T}_{a,2}$ of USIM_a (see Fig. 9(c)). The average number of traces for $k = 0$ in the second round is 4.09.

We believe that the results for the first and second rounds are different because we tuned the model architecture in Table 1 for the first round. Since we trained on the entire buffer of ChipWhisperer, traces of the first round include the computation of $\mathrm{RAND} \oplus (\mathrm{OP_C} \oplus \mathrm{K})$ in the initial round. By selecting the segments of traces more carefully and making the model's input size equal to exact size of rounds, it might be possible to get more similar results.



(a) $\mathrm{CNN}_{1,0}$ (b) $\mathrm{CNN}_{1,5}$ (c) $\mathrm{CNN}_{2,0}$

Fig. 9. Average and worst ranks for 6K traces from USIM_a.

We also tested the models' abilities to attack other byte positions. Unsurprisingly, the $\mathrm{CNN}_{1,0}$ could not recover the 5th byte of $\mathrm{OP_C} \oplus \mathrm{K}$ and vice versa, the $\mathrm{CNN}_{1,5}$ could not recover the 0th byte of $\mathrm{OP_C} \oplus \mathrm{K}$. More surprising was that the $\mathrm{CNN}_{1,0}$ showed no noticeable ability to recover the 0th subkey of RK_1. Previous work has shown that some deep learning models can do that [6]. We suppose that this is a drawback of training on the entire buffer, 24.4K data points, instead of the part of the trace corresponding to the S-box operation only, as in [6].

On the positive side, our current methodology yields highly specialized models with excellent classification accuracy. Recall that we need 300 traces from the USIM to recover the key by CPA (see Fig. 8). So, the presented models require an order of magnitude fewer traces from the victim USIM to recover the key.

5 Conclusion

We demonstrated a profiled deep learning attack on a commercial USIM card which requires less than 20 traces to recover the key (four traces on average).

Given the huge investments in deep learning, deep learning techniques are likely to become more efficient in the future. More efficient techniques may reduce the number of traces required recover the key from a USIM to a single trace. This may have serious consequences for the security of services and applications provided by mobile communication networks unless appropriate countermeasures are designed.

Acknowledgements

The authors are deeply grateful to Christophe Devine at ANSSI for providing us with re-programmable USIM cards and David Elbaze at ANSSI for helping us debug our problems with the LEIA platform. We also want to thank Huanyu Wang at KTH for helping with getting LEIA manufactured.

References

1. 3GPP TS 33.501: Security architecture and procedures for 5G system, `http://www.3gpp.org/DynaReport/33501.htm`

2. 3GPP TS 35.206: MILENAGE algorithm set: Algorithm specification, http://www.3gpp.org/DynaReport/35206.htm
3. 3GPP TS 35.231: Specification of the TUAK algorithm set, http://www.3gpp.org/DynaReport/35231.htm
4. 3GPP TS 55.205: GSM-MILENAGE algorithms: Functions a3 and a8, http://www.3gpp.org/DynaReport/55205.htm
5. Arkko, J., Norrman, K., Näslund, M., Sahlin, B.: A USIM compatible 5G AKA protocol with perfect forward secrecy. In: 2015 IEEE Trustcom/BigDataSE/ISPA. vol. 1, pp. 1205–1209 (Aug 2015)
6. Brisfors, M., Forsmark, S.: Deep-learning side-channel attacks on AES. BSc Thesis, KTH, TRITA-EECS-EX-2019:110 (2019)
7. Cagli, E., Dumas, C., Prouff, E.: Convolutional neural networks with data augmentation against jitter-based countermeasures. In: CHES. pp. 45–68 (2017)
8. Cryptography for mobile network - C implementation and Python bindings: https://github.com/mitshell/CryptoMobile
9. CW308 UFO Target: https://wiki.newae.com/CW308_UFO_Target
10. Devine, C., Pedro, M.S., Thillard, A.: A practical guide to differential power analysis of USIM cards. In: SSTIC (2018)
11. El-Baze, D., Renard, M., Trebuchet, P., Benadjila, R.: LEIA: the lab embedded ISO7816 analyzer a custom smartcard reader for the ChipWhisperer. In: SSTIC (2019)
12. Gilmore, R., Hanley, N., O'Neill, M.: Neural network based attack on a masked implementation of AES. In: HOST. pp. 106–111 (2015)
13. Goldberg, W.: GSM cloning, www.isaac.cs.berkeley.edu/isaac/gsm.html
14. GSM World: Brief history of GSM & the GSMA, http://gsmworld.com/about-us/history.htm
15. GSMA: SG.03 rules for the management and the distribution of the GSM example algorithm A3/A8 (COMP 128, COMP 128-2, and COMP128-3) used for authentication and cipher key generation v3.6
16. Hulton, D.: 3GPP 5G security (2018), http://www.dachb0den.com/projects/scard/smartcards.ppt
17. 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: ESORICS. pp. 468–480 (2015)
18. M. Briceno and I. Goldberg and D. Wagner: An implementation of the GSM A3/A8 algorithm (specifically COMP128), http://www.iol.ie/~kooltek/a3a8.txt
19. NewAE Technology: Chipwhisperer, https://newae.com/tools/chipwhisperer
20. Pahlevanzadeh, H., Dofe, J., Yu, Q.: Assessing CPA resistance of AES with different fault tolerance mechanisms. In: ASP-DAC. pp. 661–666 (Jan 2016)
21. Prasad, A.R., Zugenmaier, A., Escott, A., Soveri, M.C.: 3GPP 5G security https://www.3gpp.org/news-events/1975-sec_5g
22. Prouff, E., Strullu, R., Benadjila, R., Cagli, E., Canovas, C.: Study of deep learning techniques for side-channel analysis and introduction to ASCAD database. IACR Cryptology ePrint Archive, 2018:053 (2018)
23. SIMalliance: (2018), https://simalliance.org/wp-content/uploads/2018/11/What-is-a-3GPP-R15-5G-SIM-card-20-11-2018-FINAL-1.pdf
24. Tamas, Jos: Secrets of the SIM, http://www.hackingprojects.net/2013/04/secrets-of-sim.html
25. Welte, H.: SysmoUSIM-SJS1 user manual (2016)