Cache Side-Channel Attacks Through Electromagnetic Emanations of DRAM Accesses

Julien Maillard, Thomas Hiscock, Maxime Lecomte, and Christophe Clavier

Abstract—Remote side-channel attacks on processors exploit hardware and micro-architectural effects observable from software measurements. So far, the analysis of micro-architectural leakages over physical side-channels (power consumption, electromagnetic field) received little treatment. In this paper, we argue that those attacks are a serious threat, especially against systems such as smartphones and *Internet-of-Things* (IoT) devices which are physically exposed to the end-user. Namely, we show that the observation of *Dynamic Random Access Memory* (DRAM) accesses with an electromagnetic (EM) probe constitutes a reliable alternative to time measurements in cache side-channel attacks. We describe the EVICT+EM attack, that allows recovering a full AES key on a T-Tables implementation with similar number of encryptions than state-of-the-art EVICT+RELOAD attacks on the studied ARM platforms. This new attack paradigm removes the need for shared memory and exploits EM radiations instead of high precision timers. Then, we introduce PRIME+EM, which goal is to reverse-engineer cache usage patterns. This attack allows to recover the layout of lookup tables within the cache. Finally, we present COLLISION+EM, a collision-based attack on a *System-on-chip* (SoC) that does not require malicious code execution, and show its practical efficiency in recovering key material on an ARM TrustZone application. Those results show that physical observation of the micro-architecture can lead to improved attacks.

Index Terms—Side-Channel attack, microarchitectural attack, TrustZone, System-on-Chip

1 Introduction

MODERN *Central Processing Units* (CPUs) embed advanced prediction and optimization mechanisms to improve their performances. Several of these features, such as cache memories or speculative execution, have been shown to expose security vulnerabilities exploitable by software attacks [1]–[6]. For instance, cache-based side-channel attacks allow a malicious process to gain information about other processes, hence bypassing memory isolation provided by the *Operating System* (OS). In practice, cache attacks have been successfully employed for the recovery of cryptographic keys or application fingerprinting. These attacks have been shown to be practical on smartphones [7] as well as desktop computers [8], [9].

Embedded devices' CPUs or microcontrollers have been widely investigated through the lens of physical side-channels such as power consumption or *electromagnetic* (EM) radiations. These physical vectors have been proved to contain leakages that statistically depend on the code and data manipulated by the CPU [10]–[15]. Interestingly, smartphones embark increasingly more powerful and complex CPUs, which contain micro-architectural optimizations similar to those found in desktop computers. Smartphones are physically exposed to the users, thus falling under both micro-architectural software attacks and physical *side-channel attacks* (SCA) paradigms.

In this paper, we show that the electromagnetic emanations of *Dynamic Random Access Memory* (DRAM) accesses represent an exploitable side-channel on *Systems-on-chip* (SoC). The profiling of EM radiations has well known advantages over power consumption measurement. Particularly, it allows exploiting local leakages (coping with, for example, peripherals' noise) while being less invasive on the targeted device. We exploit this side-channel in order to perform key recovery attacks on a lookup table based cryptosystem. Moreover, the attacks presented in this paper are non-profiled: an attacker can recover secret material on a secure device without the need of prior profiling on a "whitebox" device. Because a *Last-Level Cache* (LLC) miss results in a DRAM access, this work explores variants of LLC cache attacks with physical inputs [1], [2]. The aim of this paper is to evaluate the effectiveness of DRAM access fingerprinting through EM radiations as an alternative to high precision timers. First, we design a novel attack, named EVICT+EM, an adaptation of EVICT+RELOAD [5], that has no need for shared memory with the victim and requires similar number of encryptions. Then we demonstrate a collision-based attack, COLLISION+EM, that reduces the entropy of an AES key down to 2^{68} on a SoC, and down to 2^{80} on a ARM TrustZone application in a few thousands measurements. COLLISION+EM can potentially be foreseen on recent SoCs with stacked packages, where classical physical SCAs are difficult and physical bus probing almost impossible without deteriorating the chip. Eventually, COLLISION+EM is able to recover secrets where cache flushing countermeasures are enabled, and even when the cache is partitioned and/or randomized.

- *Julien Maillard, Thomas Hiscock, and Maxime Lecomte are with University of Grenoble Alpes, CEA-Leti, MINATEC Campus, F-38054 Grenoble, France. email: firstname.name@cea.fr*
- *Christophe Clavier is with University of Limoges, XLIM-MATHIS, Limoges, France. email: firstname.name@xlim.fr*

1.1 Contributions

In this paper, we make the following contributions: (i) in section 4 we characterize DRAM accesses through EM measurements and we show that they constitute a reliable side-channel vector, (ii) we derive EVICT+EM in section 5, a hybrid attack on a T-tables AES implementation and compare the results with existing methods, (iii) in section 6 we present the PRIME+EM attack, which allows monitoring cache set accesses during encryptions or any other application, (iv) we show the practical feasibility of cache collision-based attacks with EM measurements on a high-end SoC in section 7 and (v) we apply the COLLISION+EM attack paradigm on a TrustZone application in section 8 with cache attack countermeasures and demonstrate a successful partial key recovery.

2 BACKGROUND

2.1 Physical side-channel analysis

Side-channel analysis is a specific category of physical attacks. It exploits a so called "side-channel leakage", which can lead to a disclosure of private data within the observation of auxiliary effects such as heat propagation, power consumption or EM radiation. The literature mainly studies attacks on intermediate values of cryptosystems in order to partially or fully recover sensitive data (i.e., often cryptographic keys). Depending on the attacker model, these attacks can either be *profiled* (i.e., requiring a training phase prior to the attack) or *non-profiled*.

2.2 Cache memory

SoCs embed high-speed processors that need to exchange data with "slow" DRAM. Such memories have a large storage capacity (several gigabytes), but have a high access latency. To fill the gap between CPU requirements and DRAM capacities, processor designers introduced cache memories. The smallest storage component within a cache is called a *cache line*. Cache lines are grouped within *cache ways*, that are themselves gathered into *cache sets*. When data is cached, its address (physical or virtual, depending on the architecture) is used to determine the cache set and the cache line. The cache replacement policy handles the affectation of a cache way. Different caches in a system are organized hierarchically. First level caches (L1) are fast and small, they can directly provide data to the CPU's pipeline. Upper cache levels gradually gain storage capacity at the cost of a higher response latency, until the *last level cache* (LLC) which is directly linked to the DRAM main memory, and shared by all the cores of the CPU. If the data required by the CPU are not currently in the cache, we observe a *cache miss*: the data needs to be retrieved from the higher caches (or ultimately the main memory), and the cache hierarchy is updated. On the contrary, the recovery of data already fetched in cache memory is called a *cache hit*.

2.3 Cache attacks

The ability to distinguish between cache hits and cache misses is the keystone of cache attacks. Cache attacks can be (i) time-driven if they measure the time of a complete encryption, (ii) access-driven if they analyze if target cache lines have been accessed during an encryption, (iii) trace-driven, if every memory access is profiled during an encryption. EVICT+TIME [3] and collision attacks [4] are examples of time-driven attacks. Different access-driven attacks exist, depending on the availability of cache flushing instructions (FLUSH+RELOAD, FLUSH+FLUSH) [2] or not (EVICT+RELOAD) [5]. Some attacks, such as PRIME+PROBE [3], succeed without the possession of shared memory with the victim's process. Finally, trace-driven attacks can reuse the concept of access-driven attacks, but they also require a mechanism that allows memory access timing measurements during the encryption process (e.g., process preemption techniques). There exist a myriad of variants of these attacks [16], that we leave out of the scope of this paper.

2.4 AES T-tables implementation

In this paper, we target an AES T-tables implementation from openssl-1.0.0f [17]: it is a common use-case in the literature since the work of Osvik *et al.* [3]. T-tables are precomputed lookup tables of 256×32 bits words that are designed to accelerate the computations of AES rounds. Let δ be the number of 32 bits words that can fit within a cache line. We denote by $x_i^{(r)}$ the i-th byte of the AES state at round r. Let $K_i^{(r)}$, for $0 \leq i < 4$ be the i-th 32 bits word of the key at round r (e.g., $K_0^{(r)} = (k_0^{(r)}, k_1^{(r)}, k_2^{(r)}, k_3^{(r)})$). Similarly, $W_i^{(r)}$, for $0 \leq i < 4$ represents the i-th 32 bits words of the AES state at round r (e.g., $W_0^{(r)} = (x_0^{(r)}, x_1^{(r)}, x_2^{(r)}, x_3^{(r)})$). We denote $\langle x \rangle$ the *most significant bits* (MSBs) that can be recovered thanks to a memory access observation. Namely, if $\delta = 8$, the 3 lower-bits of the T-table address cannot be recovered. In that case, $\langle x \rangle$ represent the $8-3=5$ upper bits of x. T-tables implementations consist in computing the first 9 AES rounds by consulting 4 precomputed lookup tables T_0, T_1, T_2 and T_3, as shown on Equation 1. AES state bytes for round $r' = r + 1$ are computed as follows:

$$\begin{aligned}
W_0^{(r')} &= T_0[x_0^{(r)}] \oplus T_1[x_5^{(r)}] \oplus T_2[x_{10}^{(r)}] \oplus T_3[x_{15}^{(r)}] \oplus K_0^{(r')} \\
W_1^{(r')} &= T_0[x_4^{(r)}] \oplus T_1[x_9^{(r)}] \oplus T_2[x_{14}^{(r)}] \oplus T_3[x_3^{(r)}] \oplus K_1^{(r')} \\
W_2^{(r')} &= T_0[x_8^{(r)}] \oplus T_1[x_{13}^{(r)}] \oplus T_2[x_2^{(r)}] \oplus T_3[x_7^{(r)}] \oplus K_2^{(r')} \\
W_3^{(r')} &= T_0[x_{12}^{(r)}] \oplus T_1[x_1^{(r)}] \oplus T_2[x_6^{(r)}] \oplus T_3[x_{11}^{(r)}] \oplus K_3^{(r')}
\end{aligned} \tag{1}$$

With $(x_i^{(0)})_{0 \leq i < 16}$ being the outputs of the first AddRoundKey operation (i.e., $x_i^{(0)} = p_i \oplus k_i$). The last round is computed with classical sbox substitutions. Each lookup table contains 256 elements of 32 bits each. Thanks to the C language attribute *__attribute__(aligned(·))*, we ensure in the remainder of this paper that all T-tables are aligned on $4 \times \delta$ bytes boundaries in memory, so that all tables' first element coincide with the start of a cache line: such an alignment is the worst case scenario for an attacker (misalignment effects are discussed in section 7).

3 ATTACKER MODELS

Put shortly, this work considers an attacker model that has the same requirements as traditional EM side-channel analysis. Namely, all the introduced hybrid models (*i.e.*, EVICT+EM, PRIME+EM and COLLISION+EM) require physical access to the target device, as well as a trigger signal (EM pattern, GPIO, network activity, *etc.*). Additional prerequisites of proposed attacks are highlighted in Table 1. Note that these assumptions are particularly sound in the case of smartphones that can easily be robbed.

In this paper, we use a software controlled GPIO as a trigger signal for synchronizing traces. Synchronization of traces is left out of the scope of this work, since several methods exist for this problem [18]. Also, we consider that the target device is running an algorithm that realizes secret-dependent memory accesses (instructions or data). Throughout this paper, we use the AES T-tables implementation as a meaningful use-case, with a "known plaintext" scenario, but all applications that perform memory accesses are potentially vulnerable to the attacks listed in Table 1. Finally, EVICT+EM and PRIME+EM, analogously to EVICT+RELOAD and PRIME+PROBE, require malicious code execution, while COLLISION+EM does not.

4 OBSERVING DRAM ACCESSES

In this section, we describe our experimental methodology assessing that EM radiations of DRAM accesses can be exploited as a reliable side-channel.

4.1 Device Under Test

We use a Digilent Zybo XC7Z020-1CLG400C board as our *device under test* (DUT). This board incorporates a SoC with a dual-core Cortex-A9 CPU running up to 667 MHz which belongs to the ARMv7-A family (32bit) [19]. We choose this DUT because (i) it contains a two-level cache hierarchy, (ii) the Cortex-A9 CPU contains several optimizations such as out-of-order execution, dynamic branch prediction, dual-issuing of instructions and a deep pipeline: the induced noise and jitter in EM measurements make the attack scenario realistic compared to a "smartphone context", (iii) applicative CPUs are known to have a very poor *Signal-to-Noise Ratio* (SNR) compared to simpler microcontrollers [20], [21] and Cortex-A9 on the Zybo-z7 board is no exception in this matter.

4.2 Software experimental setup

Here, we want to reliably provoke a DRAM access. The goal is to make the latter as distinguishable as possible in side-channel observations.

4.2.1 *Target code*

The target code for DRAM access discovery is depicted in Figure 1. The goal of such code snippet is to keep a constant execution while realizing a memory access whether it is a cache hit or a cache miss (i.e., DRAM access) in order to not introduce a confounding factor. It is composed of 8 steps:

- Step 1 and 8 are the function's prolog and epilog which handle the context saving (i.e., pushing and popping register values into the stack).
- Step 2 consists in initializing the `r9` register to 0: it will be used as an offset in step 4.

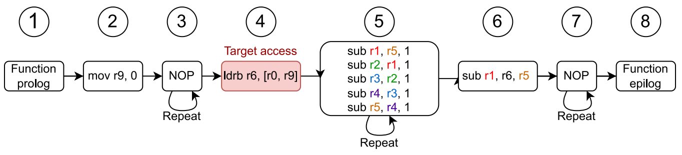


Fig. 1: Target code for DRAM access discovery.

- Step 3 and 7 operate an inline repetition of 200 `NOP` instructions. The goal of these operations is to fully flush the pipeline state and generate a visual pattern on the EM traces.
- Step 4 is the target memory access: the content at the address contained in `r0` is loaded into `r6`.
- Step 5 consists in the execution of a chainable *Read-after-Write* (RAW) dependency code snippet crafted with `sub` instructions: this forces in-order execution and single issuing. It also provides a workload to the CPU while the target memory load is processed. We chain this snippet 50 times during our experiments: this allows minimizing the total payload execution time divergences between cache hits and misses induced in step 4.
- Step 6 behaves as a synchronization barrier, as the `sub` instruction requires the `ldrb` instruction to be completed in order to use the `r6` register.

4.2.2 *Crafting eviction sets*

The access to cache maintenance instructions (such as the `clflush` instruction in x86) requires kernel privileges on ARMv7-A *Instruction Set Architecture* (ISA). Consequently, we need to craft an *eviction set* for each address we plan to target. An eviction set is a collection of addresses that fills the entire cache set in which the target address would be mapped. It is necessary for the attacker to fill the whole cache set because the cache way that would contain the target data is determined by the cache replacement policy, which is proprietary and hardly predictable. To this aim, for each targeted cache set, we select a group of congruent addresses from a large memory pool, the latter allocated through C function *mmap* with the *MAP_HUGETLB* flag activated. The congruent addresses are organised into an eviction set in the form of a double-linked list in order to tweak the hardware prefetcher: this technique is known as "pointer chasing" [3]. Once the eviction set is obtained, the eviction of a target cache line is performed by consulting each address of the eviction set.

4.2.3 *Target code wrapper*

We elaborate a controlled pre and post context around the target code execution. As we will need to discriminate if our target access is a DRAM access, the wrapper firstly implements a branchless constant time selection of a "hit target" address between the real target address A and a dummy one in order to prevent side-effect speculative behavior induced by the branch predictor (e.g., speculative loads or unwanted pipeline flushes). Then, A is evicted from the cache hierarchy by traversing an eviction set and the "hit target" is accessed: if the "hit target" is A, the target access occurring during target code execution would result

TABLE 1: Comparison of attacker models prerequisites. Attacks that are presented in this paper are indicated with ∗.

Attack	Malicious code	Shared memory	Timer	Knowledge of addresses	Physical access
EVICT+RELOAD	yes	yes	yes	yes	no
PRIME+PROBE	yes	no	yes	no	no
EVICT+EM*	yes	**no**	**no**	yes	yes
PRIME+EM*	yes	**no**	**no**	**no**	yes
COLLISION+EM*	**no**	**no**	**no**	**no**	yes

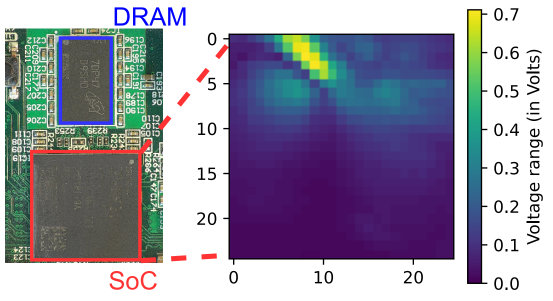


Fig. 2: Voltage amplitude cartography above the SoC.

in a cache hit, otherwise it will be a cache miss. Then, we perform a computationally intensive workload in order to force the *Digital Voltage and Frequency Scaling* (DVFS) to raise the CPU frequency to its maximum. Afterwards, a data synchronization barrier is placed to prevent late memory loads to be executed between the trigger up and the trigger down operations. Eventually, the target code is executed. We add that the whole process is tied to a single core of the chip in order to avoid context switches.

4.2.4 *Side-channel acquisition setup*

The near field EM emanations of the DUT are acquired through an EM Langer H-field RF-U 2.5-2 probe connected to a Tektronix 6 series oscilloscope with a 2.5 GHz bandwidth through a +45/50 dB low noise amplifier. The probe is attached to a 3-axis motorized bench. We use a sampling rate of 3.125 GS/s, and an *Analogic to Digital Converter* (ADC) precision of 12 bits.

4.3 Best position localization

As we observe local EM radiations, it is necessary to find an adequate probe position on the top of the chip that allows to accurately observe DRAM accesses. We expect the latter event to produce high amplitude EM radiation, because it involves the use of several components (e.g., DRAM controller, data buses, etc.). Additionally, the structure of our target code and its wrapper prevents high amplitude events, such as pipeline flushes or context switches, to occur between trigger up and trigger down events.

The amplitude of the perceived EM signal is mapped upon a 25×25 spatial grid over the main chip. Interestingly, the amplitude cartography exposes a high signal amplitude on several positions near the DRAM interconnection buses (see Figure 2). For, this DUT, we assess the best probe position as the one that captures the highest signal amplitude.

4.4 Identification of patterns

We acquire one million traces of target code execution at the best position identified previously. For each execution,

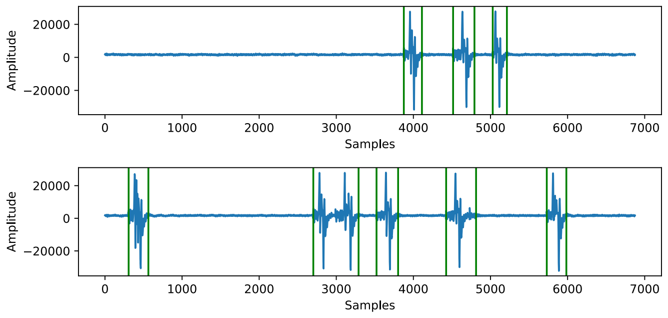


Fig. 3: High amplitude pattern identification on two example traces, green lines indicate estimated pattern boundaries.

the target access is either a cache hit or a cache miss with a 50 % probability. Several patterns emerge within the traces (see Figure 3). One can observe that (i) the patterns stand out from the remaining signal (this seems to correspond to the steps 3 and 7 baseline `NOP`s and step 5 RAW dependency instructions depicted in Figure 1) in terms of amplitude and shape, (ii) they have variable lengths, and they are located at variable positions and (iii) some of them seem unrelated to our target memory access, potentially caused by evictions from other processes. To analyze the traces, we automatically locate the patterns within the traces by applying a metric on a sliding window. More specifically, we compute the standard deviation of samples' amplitude on each window, then we establish a threshold (500 in our experiments). A standard deviation value above this threshold indicates the presence of a pattern (see Figure 3). The advantage of this method, compared to a straightforward peak detection, is that the standard deviation allows to precisely spot the boundaries of a pattern. It is also more resilient regarding the variations of the patterns' shapes, making it more generic (i.e., for different probe positions or platforms).

4.5 Lengths and locations of patterns

In this subsection, we aim at answering the following questions: (i)*Are there patterns with fixed length that appear mostly for traces where the target access is a cache miss?* (ii) *Can we relate pattern lengths to micro-architectural events?* (iii) *What can we deduce from the position of the patterns within traces?*

We start by gathering the lengths of patterns throughout our set of traces, and we label them according to cache hit or cache miss property of the corresponding target access.

In Figure 4 we observe that patterns with a width of 300 samples (approximately 110 ns) are only present in cache miss related traces: we associate the label $P1$ to such patterns. Interestingly, DRAM accesses on similar platforms last between 100 and 120 ns [22], [23]. We also observe

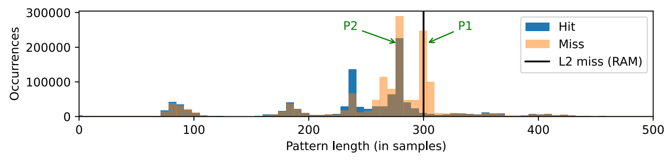


Fig. 4: Distribution of pattern lengths for hit and miss traces.

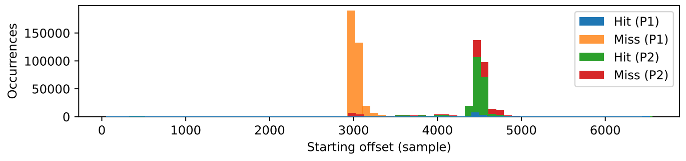


Fig. 5: Distributions of the starting offsets of patterns $P1$ and $P2$ for hit and miss related traces.

slightly shorter patterns that are present in both hit and miss related traces. We call $P2$ such patterns. These observations bring insights about the phenomena producing $P1$ and $P2$: $P1$ seems to be related to our target memory access, and $P2$ to an event independent from the target memory access, such as function epilog's `pop` instruction.

Figure 5 depicts $P1$ and $P2$ starting offset distributions within the traces. We clearly observe that $P1$ mostly appears at the middle of target code execution (i.e., where the target memory access should occur) while $P2$ occurs later. As a consequence, $P1$ seems to be the pattern corresponding to our intentional DRAM access.

4.6 Summary

In this section, we established a link between the appearance of patterns within the EM measurements with DRAM accesses. This implies that pattern matching or statistical techniques allow detecting DRAM accesses through EM radiations: this can then be used in a side-channel attack context.

5 Evict+EM

We showed in section 4 that we are able to detect DRAM accesses through EM emanations. This leads to the question: *can EM observation of DRAM accesses be exploited as an information leaking phenomenon in a cache side-channel attack context?*

This section introduces Evict+EM, a novel hybrid software and physical side-channel attack against memory accesses to recover secret keys. The attack principle is the following: (i) the attacker fills a target cache set, evicting the victim's data from the cache, (ii) the victim resumes its execution and (iii) the attacker decides whether a DRAM access has been performed or not by the victim by observing EM radiations. Note that this is an adaptation of Evict+Reload [5], with (iii) replacing the RELOAD phase. We stress that Evict+EM does not require the attacker to share memory with the victim, and hence falls into the same application contexts than Prime+Probe [3].

Algorithm 1: Target code running on the DUT.

```
Input: $P, T, L, w$
// Warmup AES encryptions
1 for $i$ from $w$ down to $0$ do
2 |   C ← AES_encrypt($P$, $K$);
3 end
4 evict($T$, $L$);
5 trigger_up();
6 C ← AES_encrypt($P$, $K$);
7 trigger_down();
```

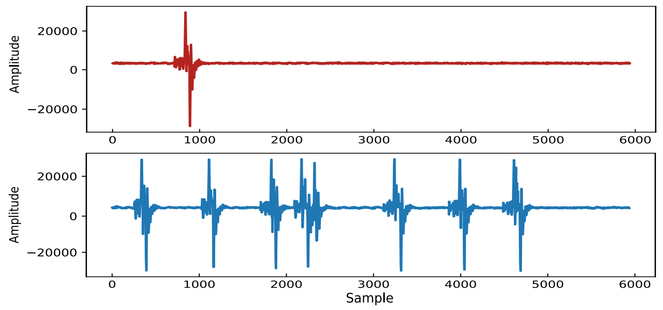


Fig. 6: EM traces of an AES encryption where the number of warmup rounds is set to 100 (top) or 0 (bottom).

5.1 Software experimental setup

In this experiment, the DUT runs a TCP server that is tied to one Cortex-A9 core. Warmup encryptions can be executed in order to cope with several jitter and noise sources. Then, one T-table related cache line, whose index is sent by the client, is evicted before the target encryption (see subsubsection 4.2.2 for the eviction procedure). This target encryption is surrounded by trigger operations.

The monitor computer samples random 16 bytes plaintexts and sends them to the DUT among several parameters such as the target T-table T, the target cache line to evict L, as well as the number of warmup rounds w. This process is repeated N times for each T-table T (see algorithm 1).

5.2 The impact of warmup encryptions

Warmup operations allow to reduce indeterminism due to cache memory, hence limiting jitter effects. When several warmup rounds are performed before evicting the target line (see Figure 6), we observe a clear single pattern within the EM measurement: the only observable DRAM access is a consequence of a cache miss during a table access in the course of the encryption (i.e., the one induced by the eviction phase). On the contrary, when no warmup rounds are performed, we observe the presence of several patterns, with different shapes, that are unrelated with the target DRAM access. Due to jitter, the target pattern that appears for 100 warmup rounds is slightly shifted in the no warmup case.

Even if performing warmup encryptions would intuitively enhance the accuracy of attacks, it would come at a cost: the attacker would be forced to wait a certain number of encryptions before triggering the target one, increasing the duration of the attack. Consequently, in subsection 5.3 we aim at designing an attack framework that is resilient regarding the noise and jitter while executing no warmup encryption.

5.3 Attack procedure

In the course of this subsection, we discuss the different steps of the key recovery attack which are (i) making hypotheses, (ii) *Region of Interest* (RoI) selection, (iii) traces preprocessing, (iv) choosing and evaluation of a metric, (v) ranking of the hypotheses according to metric value.

5.3.1 Key distinguisher

Making a hypothesis on a key byte consists in splitting the traces $(t_i)_{0\leq i\leq N}$ into two groups g_0 and g_1 based on the fact that a DRAM access at a desired encryption instant occurs or not under the hypothesis. This attack follows the methodology of the *Differential Power Analysis* (DPA) [14]. Let $k^* \in \mathbb{F}_{2^8}$ be the right key byte, $\mathcal{K} = \mathbb{F}_{2^8}$ be the set of all possible key candidates and $\mathcal{Z} = \mathbb{F}_{2^8}$ be a set of intermediate values. Then, we denote $DRAM_{\tilde{k}}(z)_{z\in\mathcal{Z}}$ a Boolean predicate that is *True* if the manipulation of z under the hypothesis $\tilde{k}$ results in a DRAM access, and *False* otherwise. Under each key hypothesis $\tilde{k}$, it is possible to separate the traces between two sets $g^0_{\tilde{k}}$ and $g^1_{\tilde{k}}$ such that $g^0_{\tilde{k}} = \{t_i \mid DRAM_{\tilde{k}}(z_i)\}$ and $g^1_{\tilde{k}} = \{t_i \mid \neg DRAM_{\tilde{k}}(z_i)\}$. Now it is possible to lift an arbitrary metric $M(g^0_{\tilde{k}}, g^1_{\tilde{k}})$, its choice is discussed in subsubsection 5.3.4. The best key hypothesis retained is then defined as:

$$k_{best} = \underset{\tilde{k}\in\mathcal{K}}{argmax}\left\{M(g^0_{\tilde{k}}, g^1_{\tilde{k}})\right\} \tag{2}$$

5.3.2 RoI and window selection

When no warmup encryption is performed, we observe (i) a jitter that misaligns our pattern of interest and (ii) other erratic patterns that appear during the encryption. Nonetheless, we make the following assumption: *It is possible to find a RoI where the interesting patterns appear more frequently for the good hypothesis.*

It is hard to specifically locate one AES round (e.g., the first) within the traces for various reasons. Firstly, the probe position does not allow us to observe pipeline EM emanations, making harder the use of *Simple Power Analysis* (SPA) in order to precisely locate the AES routines. Secondly, the presence of jitter would automatically shift the RoI.

However, guessing can be performed by knowing the algorithm. Indeed, the AES first round is unlikely to expose side-channel leakage in, say, the 10% last samples of the trace, even with the presence of jitter. In our experiments, we thus consider a RoI of 2000 samples for targeting the first AES round, which, in our experimental setup, corresponds to 400 clock cycles (approximately 640 ns).

5.3.3 Preprocessing with integral computation

As we consider discrete measurements, the integral of a trace $t = (t[j])_{0\leq j\leq n}$ is defined as the sum of its samples' values. This operation performs a linear combination of the samples over a RoI. This is useful when a jitter desynchronizes the traces. However, the main limitation of this method is that the per-sample precision is mostly lost. For the sake of our experiments, we consider integral computation upon a fixed sized sliding window of 300 samples within the RoI. This processing step is systematically applied to $g^0_{\tilde{k}}$ and $g^0_{\tilde{k}}$ before computing the metric.

5.3.4 Metric

We use the Welch's t-test as our distinguisher metric (not as a proper statistical test). It is an adaptation of Student's t-test designed to test whether two normal distributions (X_1 and X_2) have the same mean (possibly with distinct variances). This test computes a t-statistic value as follows:

$$t = \frac{\bar{E}(X_1) - \bar{E}(X_2)}{\sqrt{\frac{\bar{Var}(X_1)}{N_1} + \frac{\bar{Var}(X_2)}{N_2}}} \tag{3}$$

Here, $\bar{E}$ and $\bar{Var}$ denote the empirical mean and variance with N_1 observations of X_1 and N_2 observations of X_2. A high t-statistic indicates that the two means are highly different.

In our case, this metric seems relevant because (i) the data to be processed is divided into two groups $g^0_{\tilde{k}}$ and $g^1_{\tilde{k}}$ and (ii) the population of these two groups is very heterogeneous ($g^0_{\tilde{k}}$ has few elements): we can benefit from the in-class normalization. For the good hypothesis, we expect the t-statistic to be higher than for wrong hypotheses.

5.4 First round attack

The first step of the attack is to craft eviction sets for at least one cache set per T-table with the method described in section 4. The attacker is supposed to be able to evict at least one cache line per T-table.

The information the attacker can learn is whether one of the addresses that is mapped in this same cache line has been consulted or not. In the case of our DUT, this means that an attacker that only exploits the first AES round can only guess the five most significant bits (with $\delta = 8$) of the state bytes indexing the T-tables.

In order to attack the AES's first round, it is needed to draw hypotheses on each $(\langle x^{(0)}_i \rangle)_{0\leq i\leq 15}$. As our attacker model allows only one eviction per encryption and as each table T_j is consulted for each $(x^{(0)}_i)_{0\leq i\leq 15, i\equiv j \pmod 4}$, four sets of traces need to be gathered (one for each table). Each set of traces allow to draw hypotheses on 4 bytes. For simplicity, the targeted cache line for each table T_j corresponds to its first $\delta = 8$ elements. The global guessing entropy is obtained by performing the attack 100 times for each byte on N randomly selected traces (see Figure 7), and computing the average rank of all good hypotheses. A guessing entropy down at 0 indicates that all guesses are correct.

In Figure 7a, one can observe that the guessing entropy of EVICT+EM reaches 0 between 800 and 900 traces per table. This means that, on average, an attacker that is allowed to observe 3600 encryptions is able to guess the 5 MSBs of each byte of the secret key.

5.5 Second round attack

Following the attack of Osvik *et al.* [3], it is possible to rewrite all $(x^{(1)}_i)_{0\leq i\leq 15}$ according to the bytes of the plaintext and the secret key. Among those 16 state bytes equations, four of them are particularly interesting. Indeed, the

TABLE 2: Start of second round state target bytes, their corresponding hypotheses quadruplets to enumerate and the first round misses that need to be avoided.

Byte	Table	Key quadruplet	Round 1 misses to avoid
$x_2^{(1)}$	T_2	$(k_0, k_5, k_{10}, k_{15})$	$x_2^{(0)}, x_6^{(0)}, x_{10}^{(0)}, x_{14}^{(0)}$
$x_5^{(1)}$	T_1	(k_3, k_4, k_9, k_{14})	$x_1^{(0)}, x_5^{(0)}, x_9^{(0)}, x_{13}^{(0)}$
$x_8^{(1)}$	T_0	(k_2, k_7, k_8, k_{13})	$x_0^{(0)}, x_4^{(0)}, x_8^{(0)}, x_{12}^{(0)}$
$x_{15}^{(1)}$	T_3	$(k_1, k_6, k_{11}, k_{12})$	$x_3^{(0)}, x_7^{(0)}, x_{11}^{(0)}, x_{15}^{(0)}$

MSBs of $x_2^{(1)}$, $x_5^{(1)}$, $x_8^{(1)}$ and $x_{15}^{(1)}$ only depend on four secret key bytes LSBs (the others depend on five). Let us denote $S_i = sbox(p_i \oplus k_i)$. For the second round we can exploit the following equations:

$$\begin{aligned}
x_2^{(1)} &= S_0 \oplus S_5 \oplus 2 \cdot S_{10} \oplus 3 \cdot S_{15} \oplus sbox(k_{15}) \oplus k_2 \\
x_5^{(1)} &= S_4 \oplus 2 \cdot S_9 \oplus 3 \cdot S_{14} \oplus S_3 \oplus sbox(k_{14}) \oplus k_1 \oplus k_5 \\
x_8^{(1)} &= 2 \cdot S_8 \oplus 3 \cdot S_{13} \oplus S_2 \oplus S_7 \oplus sbox(k_{13}) \oplus k_0 \oplus k_4 \oplus k_8 \oplus 1 \\
x_{15}^{(1)} &= 3 \cdot S_{12} \oplus S_1 \oplus S_6 \oplus 2 \cdot S_{11} \oplus sbox(k_{12}) \oplus k_{15} \oplus k_3 \oplus k_7 \oplus k_{11}
\end{aligned} \tag{4}$$

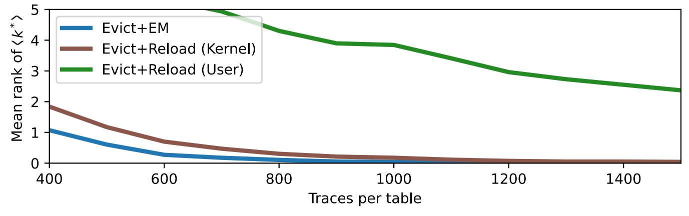
The second round attack relies on the information gained from first round attack (*i.e.*, the MSBs of each key bytes). In order to recover the full key, one needs to draw hypotheses on key bytes LSBs and use equations in Equation 4 to predict whether a DRAM access occurs for each plaintext. As each equation in Equation 4 involves only 4 key bytes, hypotheses are drawn on each quadruplet (e.g., $(k_0, k_5, k_{10}, k_{15})$ for the first equation). For each quadruplet hypothesis and target address, we select plaintexts and traces so that the target address is not accessed during the first round(see Table 2)[1]. Then, the key distinguisher (see subsubsection 5.3.1) is applied to highlight the best hypothesis for each quadruplet. Note that we are able to reuse the traces gathered for the analysis of the first round.

Globally, guessing entropies for the quadruplets converge towards 0 with less than 1600 traces per T-table on average. This means that the whole secret key can be recovered with less than 6400 traces on average.

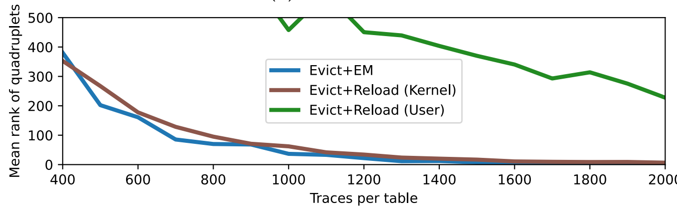
5.6 Comparison with Evict+Reload

We now compare Evict+EM with the Evict+Reload [5] attack. Note that they are similar in terms of malicious code execution, as Evict+Reload does not suppose any preemption of the victims process, nor multiple evictions per encryption. For a fair comparison, Evict+Reload uses the same eviction set construction and roaming strategies as Evict+EM. We use two different timer sources for the "Reload" part of the attack: a monotonic timer based on the *gettime* function from the *libc* denoted "User", and a high-resolution cycle counter available in ARM *Performance Monitoring Unit* (PMU). We stress that the latter requires to load a kernel module in order to allow the access to the CPU's internal performance monitoring registers: this method is ran at a kernel privilege level. Finally, we use the same distinguisher (i.e., Welch's t-test), except that our Evict+Reload attack version implements the t-test upon the timing distributions.

1. This is a slight improvement of the initial Evict+Time attack shown by Osvik *et al.* [3].



(a) First round



(b) Second round

Fig. 7: Guessing entropy comparison, per table, for Evict+Reload and Evict+EM first round attack (7a) where $\langle k^* \rangle$ is recovered, and second round attack (7b) where the full key quadruplet is recovered (see Table 2), with no warmup.

In Figure 7, we can observe that (i) Evict+EM has better performances than Evict+Reload with kernel privileges for the first round attack: this can be explained by a better temporal resolution, (ii) Evict+EM has better performances on second round quadruplets candidates, but it is less significant. An argument to explain this phenomenon is an increased amount of first round induced jitter for the Evict+EM for the second round attack. Consequently, we can assess that Evict+EM constitutes a *userland* alternative to cache attacks with similar performances than an Evict+Reload attack with kernel privileges. When performing Evict+EM, the attacker does not need to share memory with the victim, hence it is an interesting alternative to Prime+Probe family attacks when Evict+Reload is not practical.

6 Prime+EM

Evict+EM relies on the use of eviction sets to evict target cache lines: the eviction set crafting phase requires to locate the targets' physical addresses in main memory. The main goal of this section is to overcome this restriction. The technique presented in this section, called Prime+EM, is based on Prime+Probe [3]. It aims to discover the cache sets hosting the T-Tables within cache memory. This attack, which can be viewed as a reverse-engineering step, can precede an Evict+EM attack for full key recovery.

6.1 Profiling cache set activity

We assume that the T-tables are stored contiguously and 32 bytes aligned within the DRAM, and that T_0 is page aligned. For a memory page size of 4 KB, this results in the T-tables filling exactly one page so that there is no risk that two distinct T-tables share the same cache set. We also assume that the attacker knows the physical addresses that they are manipulating.

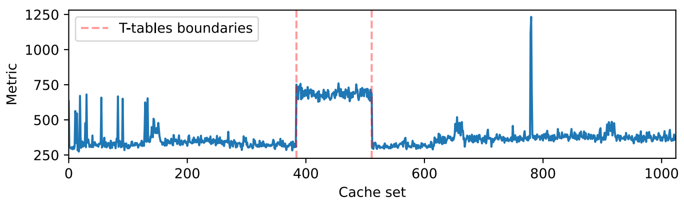


Fig. 8: Cache set activity metric of the Zybo-z7 during AES encryption with 100 warmup rounds and $N = 400$.

Then, the attacker is able to craft eviction sets for each cache set by using the methodology described in section 4. We denote ev_i the eviction set for cache set i. The profiling of cache set activity is performed as follows: (i) (Warmup) the attacker lets the victim's process perform random encryptions so as to fill the cache, (ii) (Prime) the attacker accesses ev_i to realize an eviction, (iii) (Observation) the attacker triggers the encryption of a random plaintext with the oscilloscope. This procedure is repeated several times for each cache set, and for all cache sets. For each cache set, an activity metric is computed for N EM traces as follows:

$$metric = \frac{1}{N} \sum_{k=0}^{N} \sigma_k \quad (5)$$

With σ_k being the standard deviation of the signal amplitude of the EM observation on a 1000 samples RoI for trace k. Then, an averaging of the standard deviation over the N traces is made. This metric is based on the automatic pattern identification conducted in subsection 4.4: the key idea is that DRAM accesses highly stand out compared to other activities for the assessed probe position on this DUT.

Experimental results of PRIME+EM for 100 warmup rounds with $N = 400$ are depicted in Figure 8. We clearly observe high metric values for the contiguous cache sets where the T-tables are mapped. Interestingly, we also observe non contiguous high peaks for other cache sets. These accesses can be related to cached assembly code (as we are targeting the LLC in which both instructions and data can be cached) or other memory locations accessed during the encryption (e.g., plaintext or secret key buffers). With this experiment, we show that an attacker is able to craft eviction sets targeting the T-tables without precise timers, shared memory nor knowledge of the T-tables' location in memory.

7 COLLISION+EM

The EVICT+EM and PRIME+EM attacks we presented so far have the drawback of imposing to the attacker the forgery of eviction sets. In this section, we remove the constraint of malicious code execution by designing a collision-based attack. We define a collision as equivalent to a cache hit with data that is belonging to the same target algorithm during a single encryption. Our attacker model is built under the following assumption: *every non-colliding memory access made by the victim will most likely generate a DRAM access during the targeted round*. As a prerequisite, the attacker is supposed to be able to erase T-tables' content from the cache hierarchy before the encryption. Several circumstances can validate this prerequisite, such as cache eviction from other processes, a reset of the DUT or a systematic cache flushing implemented as a cache attack countermeasure. Finally, a collision can be inferred from the absence of a DRAM access through EM measurement during the encryption. Once again, we opt for a differential approach on the EM traces.

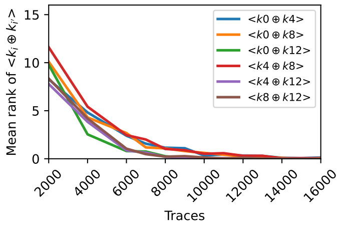
7.1 First-round attack

Let $i, j \in \{0, ..., 16\}$ with $i \neq j$ be such as $x_i^{(0)} = p_i \oplus k_i$ and $x_{i'}^{(0)} = p_{i'} \oplus k_{i'}$ are indexing the same table T. A collision is obtained when $\langle x_i^{(0)} \rangle = \langle x_{i'}^{(0)} \rangle$, which implies that $\langle p_i \oplus p_{i'} \rangle = \langle k_i \oplus k_{i'} \rangle$. Then, it is possible for an attacker to craft hypotheses on $\langle k_i \oplus k_{i'} \rangle$ under a known plaintext scenario. All the T-tables' contents are evicted from the cache before each encryption. No warmup is used, as it would fetch T-tables' contents within the cache and thwart the attack. Under each hypothesis, it is possible to separate the traces and plaintexts into two groups g^0 and g^1, based on the apparition of a collision or not for each plaintext. As in section 5, we use integral computation as preprocessing and the Welch's t-test as a distinguisher. We setup a window of 300 samples for integral computation that slides along the traces with a single sample stride.

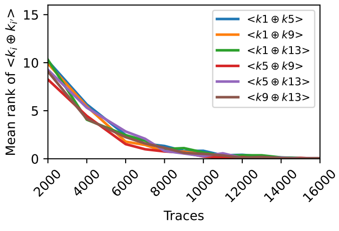
Graph theory provides an insightful representation for collision-based attacks [24]. Let $G = (V, E)$ be an undirected graph such as its vertices V represent key bytes MSBs $(\langle k_i \rangle)_{0 \leq i < 15}$. An edge is drawn between two vertices $v_i = \langle k_i \rangle$ and $v_{i'} = \langle k_{i'} \rangle$ when the knowledge of the value of v_i allows to uniquely determine the value of $v_{i'}$. Knowing a relationship of the form $\langle k_i \oplus k_{i'} \rangle = r$ creates an edge in G between v_i and $v_{i'}$, because if $\langle k_i \rangle$ is known, the value of $\langle k_{i'} \rangle$ is $\langle k_i \rangle \oplus r$. Let $G_j = (V_j, E_j)_{0 \leq j < 4}$ be subgraphs such as $V_j = \{\langle k_i \rangle \mid i \equiv j \pmod 4\}$ (see Figure 10). In other words, G_j covers key bytes that concern table T_j during the first round computations. By observing collisions within the same table, one can hope to recover enough vertices between edges of G_j to make it a connected graph.

Figure 9 illustrates the guessing entropies for each vertex corresponding to each subgraph G_j. Concerning the G_0 and G_1 subgraphs (see Figure 9a and Figure 9b), we observe that the guessing entropies converge towards 0 with between 8k and 14k attack traces. We also remark that G_2 and G_3 have a slower convergence towards 0. In our experiments, guessing entropies for each $\langle k_i \oplus k_{i'} \rangle$ reach 0 for 20k traces, making all the G_j subgraphs fully connected. This shows that our method allows discovering $\langle k_i^{(0)} \oplus k_{i'}^{(0)} \rangle$ relationships with a few thousands of encryptions on average and no malicious code running on the target.

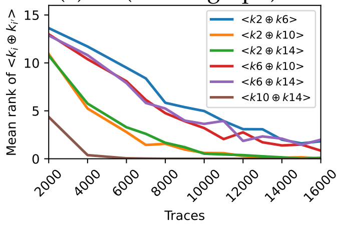
Recall that, for $\delta = 8$ (e.g., on the Zybo board), knowing if a cache line has been accessed by an intermediate value grants information upon the 5 MSBs of this value. If a subgraph G_j is connected, fixing a value for any $\langle k_i \rangle \in E_j$ allows recovering the values of all other $\langle k_{i'} \rangle \in E_j, i' \neq i$. Thus there are 2^5 hypotheses to be tested for each subgraph G_j. By combining the four subgraphs, we obtain a search space of size $2^{5\times4} = 2^{20}$. Then, remember that for $\delta = 8$, the 3 LSBs of each key byte cannot be guessed from the first round attack. Hence, after the first round attack, the total key entropy drops from 2^{128} to $2^{20} \times 2^{3\times16} = 2^{68}$.



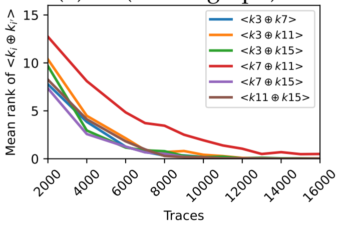
(a) T_0 (G_0 subgraph).



(b) T_1 (G_1 subgraph).



(c) T_2 (G_2 subgraph).



(d) T_3 (G_3 subgraph).

Fig. 9: Guessing entropy for G_j related collisions corresponding to each table (T_0 to T_3) on the Zybo-z7 platform, each point displays the average rank of the good candidate over 100 attack iterations upon randomly sampled traces, with no warmup.

Now we consider Equation 4. For each of the four quadruplets depicted in Table 2, each key byte involved is concerned by a different G_j. Hence, in our case, this means that the complexity of the second round attack is in the order of $4 \times 2^{20} \times 2^{3\times 4} = 2^{34}$. More precisely, an attacker would need to derive 2^{34} hypotheses in total for the four quadruplets depicted Table 2 and perform a Welch's t-test for each of them.

7.2 About connecting G

Having no connections linking the different G_j subgraphs keeps the key entropy too high to mount a bruteforce attack in a reasonable amount of time. Creating links between the subgraphs, in order to make G connected, implies that collisions must be exploitable between state bytes that are indexing different tables. Several hardware or software features could enable this, such as (i) misaligned T-tables and (ii) known or controlled data prefetching behavior.

Firstly, as stated in [25], misaligned T-tables have the effect of not mapping their base address to the beginning of a cache line. As T-tables are often contiguous in memory, this would imply that cache lines contain data from adjacent tables, and inherently allow collisions between state bytes indexing distinct tables. Secondly, data prefetching, that brings data closer to the CPU speculatively, could bring data from one table when another is accessed, potentially leading an attacker to observe a collision. Leaving aside other optimizations to the attack, a connected graph G on our DUT would lead to a total key entropy upper bound of $2^5 \times 2^{3\times 16} = 2^{53}$ after the first round attack. An upper bound of the complexity of the second round attack would be of $4 \times 2^5 \times 2^{3\times 4} = 2^{19}$ Welch's t-tests. The AES implementation we target in this experiment has no misaligned tables, and we found no exploitable prefetching behavior.

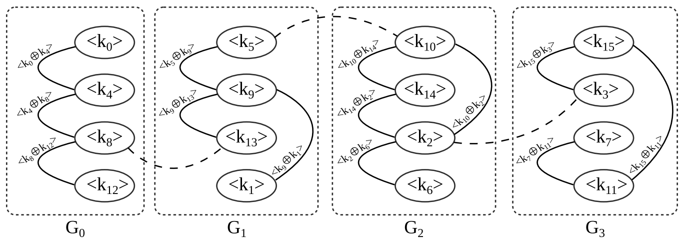


Fig. 10: Example of a collision graph G, composed of the four subgraphs G_0, G_1, G_2, G_3. Plain vertices indicate inter table collisions, dashed vertices represent the collisions that would be observable with misaligned tables or favorable prefetching behavior.

7.3 Conclusion

We were able to exploit collisions within an AES T-tables implementation with less than 20k traces on average with a non-profiled model on a Cortex-A9 processor, dropping the total key enumeration complexity from 2^{128} to 2^{68}. Putting aside the use of COLLISION+EM to reduce key entropy, this attack can be coupled to a classical EM SCA in order to reduce the total number of needed measurements. This attack has no need for malicious code execution on the DUT and only requires the attacker to flush the whole T-tables region before encryptions. Interestingly, cache flushes before sensitive application execution, which is a common countermeasure against cache attacks, would actually make COLLISION+EM possible.

8 ARM TRUSTZONE ATTACK

The ARM TrustZone is a mechanism that aims at providing hardware-based security features on ARM CPUs. ARM TrustZone ecosystems have widely been deployed in embedded devices such as smartphones, automotive and industrial systems [26]. This technology separates the so called *secure world* from the *normal world*. TrustZone provides a *Trusted Execution Environment* (TEE) which hosts the security critical features such as payment or authentication operations within *Trusted Applications* (TAs). The *secure monitor* is a privileged entity that handles context switches between secure and normal worlds. The secure and normal world's resources are isolated at the hardware-level. Such an isolation inherently prevents shared memory based attacks such as EVICT+RELOAD. For the past several years, researchers have identified several vulnerabilities in TEEs. Notably, Lipp *et al.* [7] exposed a PRIME+PROBE attack on a trusted application, bringing forward the fact that the ARM TrustZone does not guarantee "as-is" security against cache attacks. Nevertheless, the authors emphasized that some devices ensure that the cache is flushed when entering or leaving a trusted application. This countermeasure has the effect of making PRIME+PROBE attacks harder, as the eviction sets need to be browsed between the cache flushing procedure and the beginning of the encryption (respectively between the end of the encryption and the following cache flush), imposing strict timing constraints to the attacker. As a consequence, authors were unable to perform complete or partial key recovery, but rather determined if a valid or invalid key had been used by the trusted application.

The COLLISION+EM attacker model (see Table 1) is particularly relevant for targeting a TA's AES implementation with such countermeasures in a realistic scenario because (i) the addresses of the T-tables are unknown, (ii) the target cache sets are flushed before entering the secure world as a countermeasure for PRIME+PROBE, PRIME+EM and EVICT+EM attacks and (iii) no malicious code is executed (GPIO toggles before and after the encryption used to trigger the oscilloscope are not considered as malicious code), only legitimate calls to the trusted application are performed. For the rest of this section we use a STM32MP157F-EV1 dual-core Cortex-A7 based SoC with TrustZone support as our DUT. This platform encompasses several peripherals (screen, keyboard, ethernet port, etc.) that are active while performing the analysis, potentially adding extra noise and jitter to our measurements. Finally, the Cortex-A7 has two levels of cache, with a cache line size of 64 bytes and a 8-way set-associative LLC of size 1MB.

8.1 Leakage assessment

The *printed circuit board* (PCB) layout of this DUT exposes the data buses that are headed towards the two DRAM chips (see Figure 11). Similarly to subsection 4.3, we perform a 15×50 voltage range cartography during the execution of the characterization program (see Figure 1). Figure 11 illustrates that the signal amplitude is more significant above the data buses. Leakages of those buses could not be analysed on the Zybo-z7 due to the PCB layout. This is explained by the greater current through the latter. Contrary to Figure 2, the signal dynamics do not effectively highlight a best probe position: the latter needs to be determined with more specific leakage assessment. Hence, a Welch's t-test is performed (with 2000 traces per position) in order to measure the distinguishability between the traces regarding the presence of the target DRAM access. Further measurements are headed with the probe placement related to the highest t-value in Figure 11. The total procedure represents less than a day.

8.2 Attack of a trusted application

The DUT is running a full-fledged Linux distribution as a host operating system, and an OP-TEE OS [27] that handles the TrustZone environment. The OP-TEE operating system's cryptographic primitives are implemented within the LibTomCrypt library [28] whose default AES implementation is based on T-tables. Note that no cache flushing countermeasure is implemented by default before or after encryption, enabling PRIME+PROBE and PRIME+EM threat models (see Table 1). This countermeasure is hence the responsibility of the developer that writes the TA.

For the sake of this experiment, we design a trusted application that realizes T-tables AES encryptions such that (i) lookup tables' offsets are aligned to a cache line size (64 bytes on this platform) and (ii) cache lines that would hold T-tables contents are systematically evicted before and after every encryption thanks to the *TEE_cache_clean* function provided by the OP-TEE API.

To carry out this experiment, we gathered one million encryption traces. The RoI lower bound is assessed by locating the start of a region of significant activity within EM measurements. Such an instant is determined by observing an increasing trend in the average of the traces in absolute value (see Figure 12). The upper bound is determined to be empirically large so that the risk for the first round DRAM accesses to be out from the RoI is minimized.

Because of low amplitude noise and important jitter, the preprocessing method presented in subsection 5.3 is inefficient on this DUT. A variant, that counts the number of samples that exceed an amplitude threshold on a sliding window fashion, is preferred for this platform.

With a cache line size of 64 bytes, each T-table fills exactly 16 cache lines. As a consequence, a random guess on $\langle k_i \oplus k_{i'} \rangle$ ranks in eighth position in average. Also, an attacker that is able to build subgraphs G_j reduces the total AES key entropy to 2^{80}.

Guessing entropies for the first round collision attack on the four tables are depicted in Figure 13. The G_0 and G_1 subgraphs become fully connected with less than 10k traces. Some collisions are harder to detect for the G_2 and G_3 subgraphs, which become fully connected with almost 60k traces. It is important to notice that some collisions are accurately detected from 2000 traces (e.g., $\langle k_0 \oplus k_4 \rangle$). Figure 13c and Figure 13d expose that collisions with bytes belonging to $W_3^{(1)}$ need more traces to be accurately recognized. When focusing on the G_3 case, we remark that collisions related to k_6 have the worst guessing entropy convergence towards zero. Interestingly, the analysis of the binary file informed us that the compiler reordered memory accesses so that $T_2[x_6^{(0)}]$ is processed last.

8.3 Conclusion

Despite extra noise and jitter in the electromagnetic measurements, we show that COLLISION+EM succeeds in extracting secret data in a TrustZone environment. We can conclude that TrustZones with cache flushing based countermeasures are not resistant against COLLISION+EM. This raises major security concerns for TAs handling critical data (e.g., banking, authentication) on embedded devices. We are able to recover secret key material with COLLISION+EM with less than 10k encryptions. This means that information about secrets stored and manipulated by trusted environments are potentially vulnerable to COLLISION+EM with a reasonable time spent on measurements and analysis (a few hours). As mentioned, smartphones are particularly vulnerable to this threat model, as no malicious code needs to be executed on the device.

9 DISCUSSION

9.1 Comparison with EM SCA

SoCs EM radiations encompass the activity of all the components that are close to the probe. Hence, measurements of small-scale phenomena (e.g., register updates) often present a high amount of noise and jitter due to micro-architectural complexity and concurrent activity. Even so, after EM traces are gathered, traditional SCA often require preprocessing steps, such as filtering or synchronization [21]. Finally, when dealing with EM measurements on noisy SoCs, finding a good probe position is a time consuming process (i.e., several days to a month). Still, even at the best probe position,

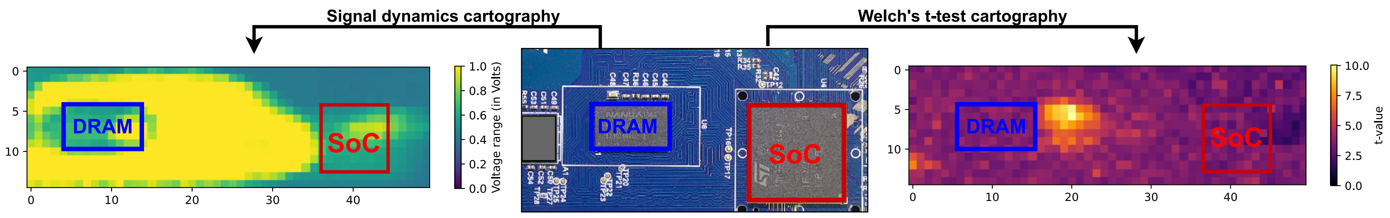


Fig. 11: Voltage range (left), top view of the board (middle) and Welch's t-test cartography (right).

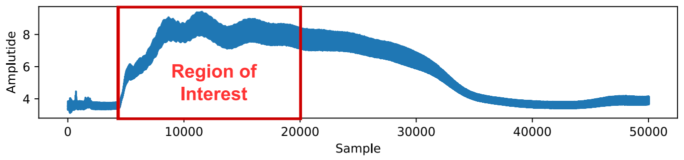
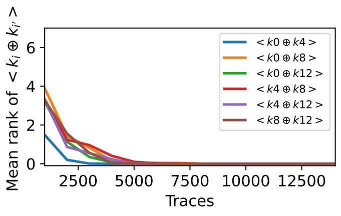
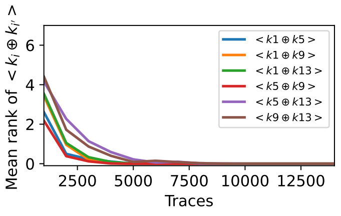


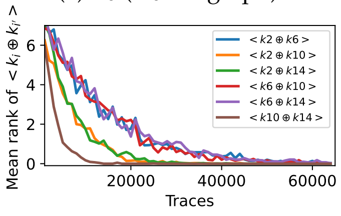
Fig. 12: Region of interest selection based on activity detection upon average of absolute traces values.



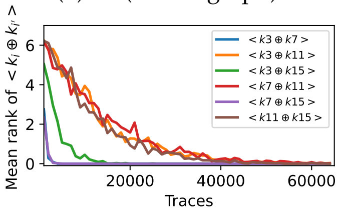
(a) T_0 (G_0 subgraph).



(b) T_1 (G_1 subgraph).



(c) T_2 (G_2 subgraph).



(d) T_3 (G_3 subgraph).

Fig. 13: Guessing entropy for G_j related collisions corresponding to each table (T_0 to T_3), each point displays the average rank of the good candidate over 100 attack iterations upon randomly sampled traces, with no warmup.

attacking a cryptosystem on a high-end SoC with non-profiled methods often requires up to millions of traces [29]. By profiling DRAM accesses through EM measurements, we tackle some of these issues. The attacks we propose are more resilient to noise and jitter than classical physical side-channel attacks. We recall that DRAM accesses last for various clock cycles and produce rather high amplitude signal: the constrains upon the acquisition chain (e.g., ADC precision, voltage precision, sampling rate, probe positioning) are less restrictive in our case. Moreover, chip packages with stacked DRAM on are known to impose a lot of EM traces gathering and preprocessing [29]. These stacked packages also limit the possibility for an attacker to directly probe the buses to harvest DRAM access information.

Combining EM measurements with the cache attack paradigm comes at the cost of being more intrusive on the DUT. As we have seen, EVICT+EM and PRIME+EM require eviction set construction, hence malicious code execution. However, COLLISION+EM removes this limitation.

9.2 Comparison with cache attacks

Cache attacks often require a high amount of malicious memory accesses. The extreme case is reached with trace-driven attacks, which need to profile almost every memory accesses performed by the victim process. Moreover, the vast majority of cache attacks require a way to measure time with enough precision. Finally, software micro-architectural attacks are often noisy. For example, access-driven and time-driven attacks on AES first round are noisy because of other rounds accesses. The method we describe here (i) has a small memory footprint, as abnormal memory interactions are performed for initial data evictions only, (ii) can be headed with better temporal precision, (iii) does not require cycle accurate timer and (iv) can target DRAM accesses through time during the encryption. In terms of drawbacks, our EVICT+EM and PRIME+EM attacks, presented in this paper, hardly allow to profile several memory addresses at the same time. Moreover, the attacker needs to have physical access on the target and add several hardware components to its experimental setup (i.e., oscilloscope, probe).

9.3 Mitigations

A natural countermeasure to EVICT+EM, PRIME+EM and COLLISION+EM is a systematic prefetch of the victim's data before encryption. This would prevent any DRAM accesses during the encryption as long as there are no self evictions occurring within the victim's process. Note that this can represent a performance bottleneck. Also, such a prefetch is not always possible, especially when the sensitive lookup tables are wider than the available cache size. A performance compromise can be reached by performing random accesses to the sensitive data before encyption: this would drastically increase the number of measurements required by an attacker. Note that this mitigation "as-is" does not prevent the attacks, and needs to be implemented jointly to other security measures (e.g., frequent rekeying).

The attacks presented in this paper exploit side-channel information leaked by DRAM accesses that are statistically dependent to a secret. Preventing table lookups and branches from depending on a secret, which is a cornerstone of so called constant time implementations [30], is also a viable mitigation strategy. Note that these methods are already widely deployed for asymmetric cryptography (e.g., square-and-multiply always for RSA).

Classical SCA countermeasures such as hiding (e.g., metallic shields or artificial EM noise addition), and masking [31] can thwart the attacks proposed in this paper. Note that, due to a high tolerance to jitter, our attacks would be poorly affected by shuffling countermeasures. Finally, concurrent activity in multi-core platforms could be employed to add noisy DRAM activity and increase the difficulty of our attacks.

9.4 Related work

Bertoni *et al.* [32] designed a simulated attack exploiting intentionally induced cache misses. More precisely, they target first round misses in a sbox AES implementation on a simulated microcontroller architecture with an 8 bytes cache line size. This attack provided inspirational insights for the EVICT+EM approach.

Osvik and al. [3] proposed an attack on an AES T-tables implementation. They formalized the key recovery procedure for attacking the two first rounds of the AES with EVICT+TIME and PRIME+PROBE. They recovered the whole key with 500k encryptions with EVICT+TIME on an AMD Athlon64 CPU. Our EVICT+EM attack is an adaptation of EVICT+TIME.

Dey *et al.* proposed a monitoring of CPU stalls induced by LLC misses through the observation of EM emanations [33]. For this purpose, they use micro-benchmarks by executing controlled code with known memory access behavior. They pinpoint the benefit of their work for benchmarking code segments when performance counters are unavailable (e.g., bootloaders). While our study and Dey *et al.*'s [33] both target similar memory events, the main purpose of our work is to mount a key recovery attack on CPUs that could be packaged with a stacked DRAM and potentially support Out-of-Order execution. Then, rather than relying on precise (and potentially device dependent) pattern matching results, we exploit statistical links between the EM radiations and DRAM accesses in order to leverage a differential non-profiled attack.

Schramm *et al.* identified that collisions of intermediate variables in a cryptographic primitive are an attack vector [34]. They experimentally confirmed this new attack paradigm by showing a chosen plaintext collision-DPA attack on the AES on an Intel 8051 compatible microcontroller [35].

Fournier and Tunstall designed a theoretical attack exploiting cache collisions during AES encryptions [36]. The target is a smartcard with a single level of cache which is shared for instruction and data. The cache line size is 16 bytes, thus containing 16 sbox elements. To break the remaining bits of each byte, they first propose a method exploiting second round SubBytes routine. Alternatively, they propose exploiting cache collisions in the MixColumns precomputed tables (i.e., the one performing the xtime function). This work has been since extended [37]. Bogdanov [24] proposed an enhanced collision-based attack targeting SubBytes and MixColumns outputs. He formalized the collision attack as a set of linear equations that can be represented as a connected graph. A practical implementation of the attack was shown on a PIC16F microcontroller. This work was extended by combining the key ranking features of DPA, *Correlation Power Analysis* (CPA) or *Mutual Information Analysis* (MIA) with collision-based attack [38].

Gérard and Standaert formalized collision attacks linear problems as a *Low Density Parity Codes* (LDPC) decoding problem [39]. They pinpoint that collision attacks are hardened by the diversity of possible implementations when considering software primitives. This work may allow to optimize the COLLISION+EM attack.

GPU caches are also vulnerable, from timing attacks [40] to correlation-collision exploiting memory coalescing [41]. Gao *et al.* investigated the EM leakages of cache collision on a NVIDIA GEFORCE GPU [42]. They find an AES key through chosen plaintext attack with 6k traces. The occurrence of a collision is used as a separation criterion for a DPA attack.

10 CONCLUSION

In this paper, we described a new methodology to exploit the electromagnetic emanations of DRAM accesses on SoCs as an attack vector. We develop three attack scenarios, EVICT+EM, PRIME+EM and COLLISION+EM, that require a physical access to the attacked device, which is relevant when considering embedded devices such as smartphones. We show that EVICT+EM enables full AES key recovery with similar precision as EVICT+RELOAD attacks with *kernel* level timer. Furthermore, the aforementioned attacks require no process interruption nor concurrency constraints. Eventually, we demonstrate the efficiency of COLLISION+EM, that do not require any malicious code execution. We showed that this technique allows partial key recovery against a T-Table AES implementation running in a trusted application on an ARM TrustZone. The attack even works with the presence of systematic cache flushing before encryptions. COLLISION+EM can be applied to a wide range of algorithms with secret dependent memory access patterns. This represents a threat regarding trusted execution environments on embedded devices, which remain physically accessible to an attacker. Future work may consider recovering the addresses of the DRAM accesses to mount more efficient attacks. Eventually, it could be beneficial to evaluate the security of mitigations suggested in subsection 9.3.

ACKNOWLEDGEMENT

This project has received funding from the European Union's Horizon Europe Grant Agreement No. 101073795 (POLIIICE).

REFERENCES

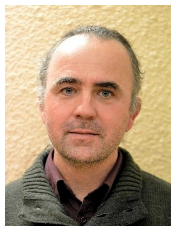
[1] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-level cache side-channel attacks are practical," in *2015 IEEE symposium on security and privacy*, pp. 605–622, IEEE, 2015.
[2] Y. Yarom and K. Falkner, "FLUSH+RELOAD: A high resolution, low noise, l3 cache side-channel attack," in *23rd USENIX security symposium*, pp. 719–732, 2014.
[3] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: the case of AES," in *Cryptographers' track at the RSA conference*, pp. 1–20, Springer, 2006.
[4] D. J. Bernstein, "Cache-timing attacks on AES," 2005.
[5] D. Gruss, R. Spreitzer, and S. Mangard, "Cache template attacks: Automating attacks on inclusive last-level caches," in *24th USENIX Security Symposium*, pp. 897–912, 2015.
[6] Y. Yarom, D. Genkin, and N. Heninger, "Cachebleed: a timing attack on openssl constant-time rsa," *Journal of Cryptographic Engineering*, 2017.
[7] M. Lipp, D. Gruss, R. Spreitzer, C. Maurice, and S. Mangard, "ARMageddon: Cache attacks on mobile devices," in *25th USENIX Security Symposium*, pp. 549–564, 2016.

[8] J. Götzfried, M. Eckert, S. Schinzel, and T. Müller, "Cache attacks on Intel SGX," in *Proceedings of the 10th European Workshop on Systems Security*, pp. 1–6, 2017.
[9] F. Brasser, U. Müller, A. Dmitrienko, K. Kostiainen, S. Capkun, and A.-R. Sadeghi, "Software grand exposure: SGX cache attacks are practical," in *USENIX Workshop on Offensive Technologies*, 2017.
[10] E. Brier, C. Clavier, and F. Olivier, "Correlation power analysis with a leakage model," in *International workshop on cryptographic hardware and embedded systems*, pp. 16–29, Springer, 2004.
[11] M. Goldack and I. C. Paar, "Side-channel based reverse engineering for microcontrollers," *Master's thesis, Ruhr-Universität Bochum, Germany*, 2008.
[12] R. Novak, "Side-channel based reverse engineering of secret algorithms," in *Proceedings of the Electrotechnical and Computer Science Conference*, 2003.
[13] C. Clavier, "Side channel analysis for reverse engineering (SCARE), an improved attack against a secret A3/A8 GSM algorithm," 2004.
[14] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Annual International Cryptology Conference*, 1999.
[15] V. Cristiani, M. Lecomte, and T. Hiscock, "A bit-level approach to side channel based disassembling," in *Conference on Smart Card Research and Advanced Applications*, Springer, 2019.
[16] Q. Ge, Y. Yarom, D. Cock, and G. Heiser, "A survey of microarchitectural timing attacks and countermeasures on contemporary hardware," *Journal of Cryptographic Engineering*, 2018.
[17] "Openssl." https://www.openssl.org.
[18] C. Fanjas, C. Gaine, D. Aboulkassimi, S. Pontié, and O. Potin, "Combined fault injection and real-time side-channel analysis for android secure-boot bypassing," in *International Conference on Smart Card Research and Advanced Applications*, pp. 25–44, Springer, 2022.
[19] A. LTD, "Cortex-a9 technical reference manual," *Revision: r4p1*, 2012.
[20] J. Maillard, T. Hiscock, M. Lecomte, and C. Clavier, "Side-channel disassembly on a system-on-chip: A practical feasibility study," *Microprocessors and Microsystems*, vol. 101, p. 104904, 2023.
[21] J. Longo, E. D. Mulder, D. Page, and M. Tunstall, "Soc it to em: electromagnetic side-channel attacks on a complex system-on-chip," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 620–640, Springer, 2015.
[22] "7-cpu." https://7-cpu.com/cpu/Cortex-A9.html.
[23] "Zynq benchmarks." "https://www.jblopen.com/zynq-benchmarks/".
[24] A. Bogdanov, "Improved side-channel collision attacks on AES," in *International Workshop on Selected Areas in Cryptography*, pp. 84–95, Springer, 2007.
[25] R. Spreitzer and T. Plos, "Cache-access pattern attack on disaligned AES T-tables," in *Workshop on Constructive Side-Channel Analysis and Secure Design*, pp. 200–214, Springer, 2013.
[26] S. Pinto and N. Santos, "Demystifying arm trustzone: A comprehensive survey," *ACM computing surveys (CSUR)*, vol. 51, no. 6, 2019.
[27] "OP-TEE." https://www.op-tee.org/.
[28] "LibTomCrypt." https://www.libtom.net/LibTomCrypt/.
[29] O. Lisovets, D. Knichel, T. Moos, and A. Moradi, "Let's take it offline: Boosting brute-force attacks on iphone's user authentication through sca," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 496–519, 2021.
[30] J. B. Almeida, M. Barbosa, G. Barthe, F. Dupressoir, and M. Emmi, "Verifying {Constant-Time} implementations," in *25th USENIX Security Symposium (USENIX Security 16)*, pp. 53–70, 2016.
[31] S. Mangard, E. Oswald, and T. Popp, *Power analysis attacks: Revealing the secrets of smart cards*. Springer, 2008.
[32] G. Bertoni, V. Zaccaria, L. Breveglieri, M. Monchiero, and G. Palermo, "AES power attack based on induced cache miss and countermeasure," vol. 1, pp. 586– 591 Vol. 1, 05 2005.
[33] M. Dey, A. Nazari, A. Zajic, and M. Prvulovic, "Emprof: Memory profiling via em-emanation in iot and hand-held devices," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 881–893, IEEE, 2018.
[34] K. Schramm, T. Wollinger, and C. Paar, "A new class of collision attacks and its application to DES," in *International Workshop on Fast Software Encryption*, pp. 206–222, Springer, 2003.
[35] K. Schramm, G. Leander, P. Felke, and C. Paar, "A collision-attack on AES combining side channel and differential-attack.," *Submitted for Publication*, 2003.
[36] J. Fournier and M. Tunstall, "Cache based power analysis attacks on AES," in *Australasian Conference on Information Security and Privacy*, pp. 17–28, Springer, 2006.
[37] J.-F. Gallais, I. Kizhvatov, and M. Tunstall, "Improved trace-driven cache-collision attacks against embedded aes implementations," in *International Workshop on Information Security Applications*, pp. 243–257, Springer, 2010.
[38] A. Bogdanov and I. Kizhvatov, "Beyond the limits of DPA: combined side-channel collision attacks," *IEEE Transactions on Computers*, vol. 61, no. 8, pp. 1153–1164, 2011.
[39] B. Gérard and F.-X. Standaert, "Unified and optimized linear collision attacks and their application in a non-profiled setting: extended version," *Journal of Cryptographic Engineering*, vol. 3, no. 1, pp. 45–58, 2013.
[40] Z. H. Jiang, Y. Fei, and D. Kaeli, "A complete key recovery timing attack on a gpu," in *IEEE International symposium on high performance computer architecture (HPCA)*, pp. 394–405, IEEE, 2016.
[41] J. Ahn, C. Jin, J. Kim, M. Rhu, Y. Fei, D. Kaeli, and J. Kim, "Trident: A hybrid correlation-collision GPU cache timing attack for AES key recovery," in *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 332–344, IEEE, 2021.
[42] Y. Gao, W. Cheng, H. Zhang, and Y. Zhou, "Cache-collision attacks on GPU-based AES implementation with electro-magnetic leakages," in *Conference On Trust, Security And Privacy In Computing And Communications*, pp. 300–306, IEEE, 2018.

Julien Maillard is a PhD student at the french Commissariat à l'énergie Atomique et aux energies alternatives (CEA) and affiliated to the University of Limoges, France. He obtained his master's degree in computer science and cryptology at University of Limoges, France. His main research topics include side-channel analysis, reverse-engineering of memory components as well as micro-architectural attacks.

Thomas Hiscock is a researcher at the french Commissariat à l'énergie Atomique et aux energies alternatives (CEA). He obtained his PhD in 2017 in computer science. He performs various security testing on embedded devices with CEA's industrial partners. His main research topics include the design of secure processors (against hardware attacks), side-channel analysis, fault-injections as well as micro-architectural attacks.

Maxime Lecomte received the Ph.D. degree in microelectronics, in 2016. He is a Researcher with the French Commissariat à l'énergie Atomique et aux Énergies Alternatives (CEA). His main research interests include side-channel analysis, fault injections as well as physical unclonable functions, and true random generators.



Christophe Clavier has been graduated in 1991 from the Engineering School of Telecommunication in Paris. He has worked in Gemplus from 1998 to 2008 where he was in charge of smartcards security against fault and side-channel threats. From 2010 on he is Professor at University of Limoges.