



Cache Side-Channel Attacks Through Electromagnetic Emanations of DRAM Accesses

Julien Maillard^{1,2}^a, Thomas Hiscock¹^b, Maxime Lecomte¹^c and Christophe Clavier²^d

¹Univ. Grenoble Alpes, CEA-LETI, Minatec Campus, f-38054 Grenoble, France

²Univ. Limoges, XLIM-MATHIS, Limoges, France


Keywords: Side-Channel Attack, Micro-Architectural Attack, TrustZone, System-on-Chip.


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 of applications. 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.


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 (Liu et al., 2015; Yarom and Falkner, 2014; Osvik et al., 2006; Bernstein, 2005; Gruss et al., 2015; Yarom et al., 2017). For instance, cache-based *Side-Channel Attacks* (SCA) 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 (Lipp

et al., 2016) as well as desktop computers (Götzfried et al., 2017; Brasser et al., 2017). Embedded devices' CPUs or microcontrollers have been widely investigated through the lens of physical side-channels such as power consumption or *Electromagnetic* (EM) radiations. The behavior of these physical vectors have been shown to be statistically depend on the code and data manipulated by the CPU (Brier et al., 2004; Goldack and Paar, 2008; Novak, 2003; Clavier, 2004; Kocher et al., 1999; Cristiani et al., 2019). 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 the scope of both micro-architectural software attacks and physical side-channel attacks. In this paper, we show that the electromagnetic emanations of *Dynamic Random Access Memory* (DRAM) accesses represent an exploitable side-channel on a *Systems-on-chip* (SoC). The profiling of EM radiations has well known advantages over power consumption measurement. Par-

^a <https://orcid.org/0009-0002-2267-7621>

^b <https://orcid.org/0009-0001-5183-2291>

^c <https://orcid.org/0000-0002-9985-7586>

^d <https://orcid.org/0000-0002-0767-3684>

ticularly, 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 hybrid approaches of LLC cache attacks with physical inputs (Liu et al., 2015; Yarom and Falkner, 2014).

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 EVICT+RELOAD (Gruss et al., 2015), (iii) in section 6 we present the PRIME+EM attack, which allows monitoring cache of a victim process, (iv) we show the practical feasibility of cache collision-based attacks with EM measurements on a 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 (Osvik et al., 2006) and collision attacks (Bernstein, 2005) are examples of time-driven attacks. Different access-driven attacks exist, depending on the availability of cache flushing instructions (FLUSH+RELOAD, FLUSH+FLUSH) (Yarom and Falkner, 2014) or not (EVICT+RELOAD) (Gruss et al., 2015). Some attacks, such as PRIME+PROBE (Osvik et al., 2006), 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 (Ge et al., 2018), 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: it is a common use-case in the literature since the work of Osvik *et al.* (Osvik

et al., 2006). 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-bit 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-bit 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-bit 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:

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

With $(x_i^{(0)})_{0 \leq i < 16}$ being the outputs of the first AdRoundKey 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. In our target software, we ensure in the remainder of this paper that all T-Tables are aligned on the beginning 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 attacks (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. We use a software controlled GPIO as a trigger signal for synchronizing traces. Note that the literature exposes several ways of synchronizing measurements without GPIO triggers (Fanjias et al., 2022), but we leave this out the scope of this paper. 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, similarly 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 (32-bit) (LTD, 2012). We choose this DUT because (i) it contains a two-level cache hierarchy, (ii) the Cortex-A9 CPU implements 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 (Mailard et al., 2023; Longo et al., 2015) and Cortex-A9 on the Zybo-z7 board is no exception in this matter.

4.2 Software Experimental Setup

We aim at reliably provoking a DRAM access that is as distinguishable as possible in side-channel observations. Target code for DRAM access discovery is depicted in Figure 1. The goal of this code snippet is to minimize execution time variations while realizing a memory access, whether it is a cache hit or a cache miss (i.e., DRAM access). 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. Step 3 and 7 operate an inline repetition of `NOP` instructions to fully flush the pipeline state and generate a visual pattern on the EM traces. Then, step 4 performs the target memory

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

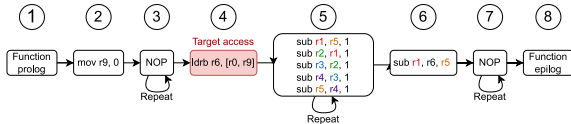


Figure 1: Target code for DRAM access discovery.

access. Step 5 consists in the execution of a *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 divergences in the execution time of the full code snippet 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.

4.2.1 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” (Osvik et al., 2006). Once the eviction set is obtained, the eviction of a target cache line is performed by consulting each address of the eviction set.

4.2.2 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

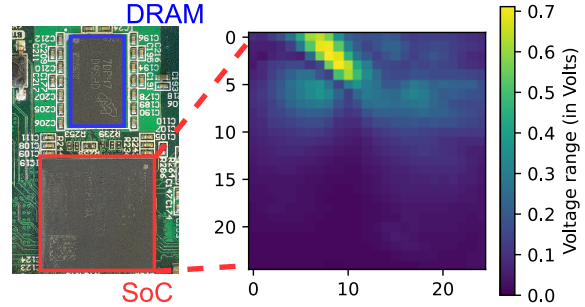


Figure 2: Voltage amplitude cartography above the SoC.

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 Probe Location

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 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, 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 ob-

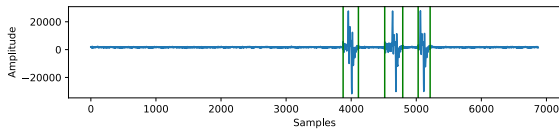


Figure 3: Identification of 3 different patterns, green lines indicate estimated pattern boundaries. Pattern at offset 4000 corresponds to the target DRAM access.

serve that some patterns stand out from the remaining signal in terms of amplitude and shape. During our experiments on this test case, we noticed that DRAM accesses increased the probability to observe a pattern in the region close to time sample 4000, oppositely to measurements realized for cache hits. Note that other patterns appear erratically (see Figure 3), but those seem unrelated to our target memory access, potentially caused by evictions from other processes. This implies that pattern matching or statistical techniques allow the detection 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, with step (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 (Osvik et al., 2006).

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 subsection 4.2.1 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 .

5.2 Attack Procedure

5.2.1 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) (Kocher et al., 1999). 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 $g_{\tilde{k}}^0$ and $g_{\tilde{k}}^1$ such that $g_{\tilde{k}}^0 = \{t_i \mid DRAM_{\tilde{k}}(z_i)\}$ and $g_{\tilde{k}}^1 = \{t_i \mid \neg DRAM_{\tilde{k}}(z_i)\}$. Considering an arbitrary metric M , the best key hypothesis retained is then defined as:

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

We use the Welch’s t-test as our distinguisher metric (not as a 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). In our case, this metric seems relevant because (i) the data to be processed is divided into two groups $g_{\tilde{k}}^0$ and $g_{\tilde{k}}^1$ and (ii) the population of these two groups is very heterogeneous ($g_{\tilde{k}}^0$ 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.

5.2.2 RoI Selection and Preprocessing

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 EM emanations from the CPU’s pipeline, making harder the use of *Simple Power Analysis* (SPA) in order to precisely locate the AES routines. However, guessing can be performed when the underlying algorithm is known. 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 time samples, that corresponds to 400 clock cycles (approximately 640 ns). 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_k^0 and g_k^1 before computing the metric.

5.3 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 $(\langle x_i^{(0)} \rangle)_{0 \leq i \leq 15}$. As our attacker model allows only one eviction per encryption and as each table T_j is consulted for each $(x_i^{(0)})_{0 \leq i \leq 15, i \equiv j \pmod{4}}$, 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 4), and computing the average rank of all good hypotheses. A guessing entropy down at 0 indicates that all guesses are correct. In Figure 4a, 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.

5.4 Second Round Attack

Following the attack of Osvik *et al.* (Osvik *et al.*, 2006), it is possible to rewrite all $(x_i^{(1)})_{0 \leq i \leq 15}$ 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 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 $S_i = \text{sbox}(p_i \oplus k_i)$. For the second round we can exploit the following equations:

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

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 3 to predict whether a DRAM access occurs for each plaintext. As each equation in Equation 3 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. Note that this is a slight improvement of the initial EVICT+TIME attack shown by Osvik *et al.* (Osvik *et al.*, 2006). Then, the key distinguisher (see subsection 5.2.1) 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 (see Figure 4b). This means that the whole secret key can be recovered with less than 6400 traces on average.

5.5 Comparison with EVICT+RELOAD

We now compare EVICT+EM with the EVICT+RELOAD (Gruss *et al.*, 2015) 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

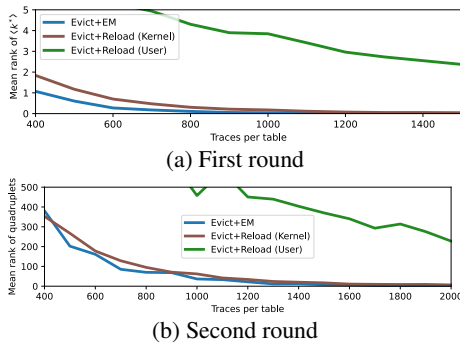


Figure 4: Guessing entropy comparison, averaged per per table, for EVICT+RELOAD and EVICT+EM first round attack (4a) where $\langle k^* \rangle$ is recovered, and second round attack (4b) where the full key quadruplet is recovered, with no warmup.

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. In Figure 4, 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. Finally, 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 (Osvik et al., 2006). Here, we monitor cache activity in order to identify 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.

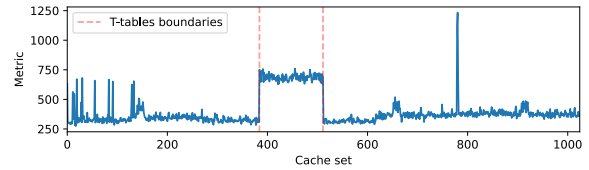


Figure 5: Cache set activity metric of the Zybo-z7 during AES encryption with 100 warmup rounds and $N = 400$.

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. Then, the attacker is able to craft eviction sets for each cache set within the cache 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:

$$metric = \frac{1}{N} \sum_{k=0}^N \sigma_k \quad (4)$$

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 (i.e., 100 encryptions of the same plaintext before rising the GPIO) with $N = 400$ are depicted in Figure 5. 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

EVICT+EM and PRIME+EM require 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 assumption that all non-colliding memory accesses made by the victim would 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. This scenario can be enabled by cache eviction from other processes or by 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 $i, j \in \{0, \dots, 16\}$ with $i \neq j$ be such as $x_i^{(0)} = p_i \oplus k_i$ and $x_j^{(0)} = p_j \oplus k_j$ are indexing the same table T . A collision is obtained when $\langle x_i^{(0)} \rangle = \langle x_j^{(0)} \rangle$, which implies that $\langle p_i \oplus p_j \rangle = \langle k_i \oplus k_j \rangle$. Then, an attacker can craft hypotheses on $\langle k_i \oplus k_j \rangle$ under a known plaintext scenario. All the T-Tables' contents are evicted from the cache before each encryption. 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. Graph theory provides an insightful representation for collision-based attacks (Bogdanov, 2007). 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 $v_j = \langle k_j \rangle$ when the knowledge of the value of v_i allows to uniquely determine the value of v_j . Knowing a relationship of the form $\langle k_i \oplus k_j \rangle = r$ creates an edge in G between v_i and v_j , because if $\langle k_i \rangle$ is known, the value of $\langle k_j \rangle$ 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 7). 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 6 illustrates the guessing entropies for each vertex corresponding to each subgraph G_j . Concerning the G_0 and G_1 subgraphs (see Figure 6a and Figure 6b), we observe that the guessing entropies converge towards 0 with between 8k and 14k attack traces. We also remark that G_2 and

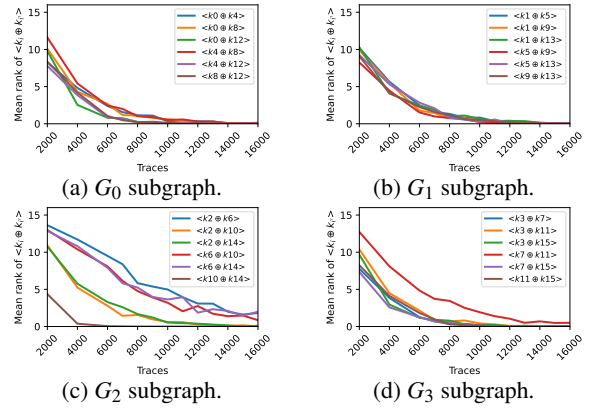


Figure 6: Guessing entropy for G_j related collisions corresponding to each table (T_0 to T_3) on the Zybo-z7 platform, each point shows the average good candidate rank averaged over 100 attacks.

G_3 have a slower convergence towards 0. In our experiments, guessing entropies for each $\langle k_i \oplus k_j \rangle$ reach 0 for 20k traces, making all the G_j subgraphs fully connected. This shows that our method allows discovering $\langle k_i^{(0)} \oplus k_j^{(0)} \rangle$ relationships with a few thousands of encryptions on average. 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 $\langle k_j \rangle \in E_j, i' \neq i$. 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 $2^{5 \times 4} = 2^{20}$. 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 $2^{20} \times 2^{3 \times 16} = 2^{68}$. Now we consider Equation 3. For each of the four quadruplets, 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 and perform a Welch's t-test for each of them.

7.2 About Connecting G

Creating links between subgraphs 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 (Spreitzer and Plos, 2013), misaligned T-Tables have the effect of not mapping their base address to the beginning of a cache line. As T-Tables are often

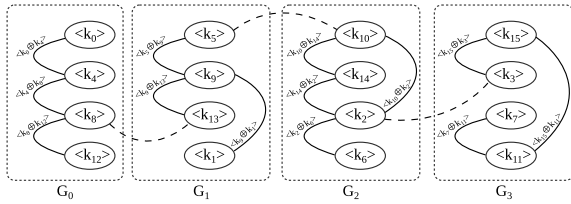


Figure 7: Example of collision graph G composed of four subgraphs G_0 to G_3 . Plain vertices indicate inter table collisions, dashed vertices represent collisions observable with misaligned tables or favorable prefetching behavior.

contiguous in memory, this would imply that cache lines contain data from adjacent tables, allowing 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. The AES implementation we target here has no misaligned tables, and we found no exploitable prefetching behavior.

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 (Pinto and Santos, 2019). 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 isolation 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.* (Lipp et al., 2016) exposed a PRIME+PROBE attack on a trusted application. Note that their attack only allow to classify valid versus wrong keys. The authors emphasized that some devices ensure that the cache is flushed when entering or leaving a trusted application. This countermeasure make PRIME+PROBE attacks harder, as eviction sets need to be browsed between the cache flushing procedure and the beginning of the encryption, imposing precise synchronization. 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 applica-

tion. The COLLISION+EM attacker model 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. Here, 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 during the analysis, adding 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 1Mb 8-way set-associative LLC.

8.1 Attack of a Trusted Application

The DUT is running a full-fledged Linux distribution as a host operating system, and an OP-TEE OS that handles the TrustZone environment. OP-TEE’s cryptographic primitives are implemented within the LibTomCrypt library, with T-Tables AES implementation by default. 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 AES encryptions, where 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. Because of low amplitude noise and important jitter, the preprocessing method presented in subsection 5.2 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 $\langle k_i \oplus k_j \rangle$ 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 8. 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

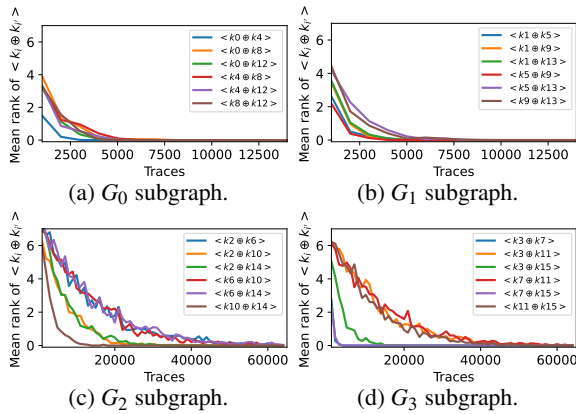


Figure 8: Guessing entropy for G_j related collisions corresponding to each table (T_0 to T_3), each point shows the good candidate rank averaged over 100 attacks.

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 8c and Figure 8d 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 see that collisions related to k_6 have the worst guessing entropy convergence towards zero. Interestingly, the analysis of the binary file indicates that the compiler reordered memory accesses so that $T_2[x_6^{(0)}]$ is processed last. Despite extra noise and jitter in the electromagnetic measurements, COLLISION+EM succeeds in extracting secret data in a TrustZone environment. Hence, TrustZones with cache flushing countermeasures are not resistant against COLLISION+EM. Moreover, as COLLISION+EM does not rely on eviction sets, cache partitioning or randomization seem inefficient.

9 DISCUSSION

9.1 Comparison with EM SCA

On a SoC, measurements of small-scale phenomena through EM (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 (Longo et al., 2015). Finally, finding a good probe position on noisy SoCs is a time consuming process (i.e., several days to a month). Still, even at the best probe position, attacking a cryptosystem on a high-end SoC with non-profiled methods often requires up to millions of traces (Lisovets et al., 2021). 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 constraints upon the acquisition chain (e.g., ADC precision, sampling rate, etc.) 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 (Lisovets et al., 2021). 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. EVICT+EM and PRIME+EM require eviction set construction, hence malicious code execution, but 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, EVICT+EM and PRIME+EM attacks 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 the presented attacks is a systematic prefetch of the victim’s data before encryption. This would prevent DRAM accesses during the encryption supposing that no self evictions occur 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

encryption: 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 secret dependent table lookups and branches, which is a cornerstone of constant time implementations (Almeida et al., 2016), is also a viable mitigation strategy 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 (Mangard et al., 2008) can thwart the attacks proposed in this paper. Because of a high tolerance to jitter, our attacks would be poorly affected by shuffling countermeasures.

9.4 Related Work

Bertoni *et al.* (Bertoni et al., 2005) designed a simulated attack exploiting intentional cache misses. They targeted the first round of a sbox AES implementation on a simulated microcontroller architecture with an 8 bytes cache line size. This provided inspirational insights for the EVICT+EM approach. Osvik and al. (Osvik et al., 2006) proposed cache attacks on an AES T-Tables implementation AMD Athlon64 CPU. They formalized the key recovery procedure to attack the two first rounds of the AES with EVICT+TIME and PRIME+PROBE. Dey *et al.* proposed to monitor CPU stalls induced by LLC misses through EM emanations (Dey et al., 2018). They used 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 both target similar memory events, our goal 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 to leverage a differential non-profiled attack. Schramm *et al.* identified that collisions of intermediate variables in a cryptographic primitive are an attack vector (Schramm et al., 2003b; Schramm et al., 2003a). Fournier and Tunstall designed a theoretical attack exploiting cache collisions during AES encryptions (Fournier and Tunstall, 2006) on a smartcard with a single

level of cache. They propose exploiting cache collisions in the MixColumns pre-computed tables. This work has been since extended (Gallais et al., 2010). Bogdanov (Bogdanov, 2007) 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. 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 (Bogdanov and Kizhvatov, 2011). Gérard and Standaert formalized collision attacks linear problems as a *Low Density Parity Codes* (LDPC) decoding problem (Gérard and Standaert, 2013). They pinpoint that collision attacks are hardened by the diversity of possible implementations when considering software primitives. This work may allow to optimize COLLISION+EM.

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. Then, we demonstrate the efficiency of COLLISION+EM, that do not require any malicious code execution, and allows partial key recovery against a T-Table AES implementation running in a trusted application on an ARM TrustZone, even with cache flushing countermeasure activated. Besides, randomized and partitioned caches seem inefficient against COLLISION+EM. This threatens 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. Finally, it could be beneficial to evaluate the efficiency of mitigations suggested in subsection 9.3.

ACKNOWLEDGEMENTS

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

REFERENCES

- Almeida, J. B., Barbosa, M., Barthe, G., Dupressoir, F., and Emmi, M. (2016). Verifying {Constant-Time} implementations. In *25th USENIX Security Symposium (USENIX Security 16)*.
- Bernstein, D. J. (2005). Cache-timing attacks on AES.
- Bertoni, G., Zaccaria, V., Brevoglieri, L., Monchiero, M., and Palermo, G. (2005). AES power attack based on induced cache miss and countermeasure. volume 1.
- Bogdanov, A. (2007). Improved side-channel collision attacks on AES. In *International Workshop on Selected Areas in Cryptography*.
- Bogdanov, A. and Kizhvatov, I. (2011). Beyond the limits of DPA: combined side-channel collision attacks. *IEEE Transactions on Computers*, 61(8).
- Brasser, F., Müller, U., Dmitrienko, A., Kostianen, K., Capkun, S., and Sadeghi, A.-R. (2017). Software grand exposure: SGX cache attacks are practical. In *USENIX Workshop on Offensive Technologies*.
- Brier, E., Clavier, C., and Olivier, F. (2004). Correlation power analysis with a leakage model. In *International workshop on cryptographic hardware and embedded systems*.
- Clavier, C. (2004). Side channel analysis for reverse engineering (SCARE), an improved attack against a secret A3/A8 GSM algorithm.
- Cristiani, V., Lecomte, M., and Hiscock, T. (2019). A bit-level approach to side channel based disassembling. In *Conference on Smart Card Research and Advanced Applications*.
- Dey, M., Nazari, A., Zajic, A., and Prvulovic, M. (2018). Emprof: Memory profiling via em-emanation in iot and hand-held devices. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*.
- Fanjas, C., Gaine, C., Aboukassimi, D., Pontié, S., and Potin, O. (2022). Combined fault injection and real-time side-channel analysis for android secure-boot bypassing. In *International Conference on Smart Card Research and Advanced Applications*.
- Fournier, J. and Tunstall, M. (2006). Cache based power analysis attacks on AES. In *Australasian Conference on Information Security and Privacy*.
- Gallais, J.-F., Kizhvatov, I., and Tunstall, M. (2010). Improved trace-driven cache-collision attacks against embedded aes implementations. In *International Workshop on Information Security Applications*.
- Ge, Q., Yarom, Y., Cock, D., and Heiser, G. (2018). A survey of microarchitectural timing attacks and countermeasures on contemporary hardware. *Journal of Cryptographic Engineering*.
- Gérard, B. and Standaert, F.-X. (2013). Unified and optimized linear collision attacks and their application in a non-profiled setting: extended version. *Journal of Cryptographic Engineering*, 3(1).
- Goldack, M. and Paar, I. C. (2008). Side-channel based reverse engineering for microcontrollers. *Master's thesis, Ruhr-Universität Bochum, Germany*.
- Götzfried, J., Eckert, M., Schinzel, S., and Müller, T. (2017). Cache attacks on Intel SGX. In *Proceedings of the 10th European Workshop on Systems Security*.
- Gruss, D., Spreitzer, R., and Mangard, S. (2015). Cache template attacks: Automating attacks on inclusive last-level caches. In *24th USENIX Security Symposium*.
- Kocher, P., Jaffe, J., and Jun, B. (1999). Differential power analysis. In *Annual International Cryptology Conference*.
- Lipp, M., Gruss, D., Spreitzer, R., Maurice, C., and Mangard, S. (2016). ARMageddon: Cache attacks on mobile devices. In *25th USENIX Security Symposium*.
- Lisovets, O., Knichel, D., Moos, T., and Moradi, A. (2021). Let's take it offline: Boosting brute-force attacks on iphone's user authentication through sca. *IACR Transactions on Cryptographic Hardware and Embedded Systems*.
- Liu, F., Yarom, Y., Ge, Q., Heiser, G., and Lee, R. B. (2015). Last-level cache side-channel attacks are practical. In *2015 IEEE symposium on security and privacy*.
- Longo, J., Mulder, E. D., Page, D., and Tunstall, M. (2015). Soc it to em: electromagnetic side-channel attacks on a complex system-on-chip. In *International Workshop on Cryptographic Hardware and Embedded Systems*.
- LTD, A. (2012). Cortex-a9 technical reference manual. *Revision: r4p1*.
- Maillard, J., Hiscock, T., Lecomte, M., and Clavier, C. (2023). Side-channel disassembly on a system-on-chip: A practical feasibility study. *Microprocessors and Microsystems*, 101.
- Mangard, S., Oswald, E., and Popp, T. (2008). *Power analysis attacks: Revealing the secrets of smart cards*. Springer.
- Novak, R. (2003). Side-channel based reverse engineering of secret algorithms. In *Proceedings of the Electrotechnical and Computer Science Conference*.
- Osvik, D. A., Shamir, A., and Tromer, E. (2006). Cache attacks and countermeasures: the case of AES. In *Cryptographers' track at the RSA conference*.
- Pinto, S. and Santos, N. (2019). Demystifying arm trust-zone: A comprehensive survey. *ACM computing surveys (CSUR)*, 51(6).
- Schramm, K., Leander, G., Felke, P., and Paar, C. (2003a). A collision-attack on AES combining side channel and differential-attack. *Submitted for Publication*.
- Schramm, K., Wollinger, T., and Paar, C. (2003b). A new class of collision attacks and its application to DES. In *International Workshop on Fast Software Encryption*.
- Spreitzer, R. and Plos, T. (2013). Cache-access pattern attack on disaligned AES T-tables. In *Workshop on Constructive Side-Channel Analysis and Secure Design*.
- Yarom, Y. and Falkner, K. (2014). FLUSH+RELOAD: A high resolution, low noise, l3 cache side-channel attack. In *23rd USENIX security symposium*.
- Yarom, Y., Genkin, D., and Heninger, N. (2017). Cachebleed: a timing attack on openssl constant-time rsa. *Journal of Cryptographic Engineering*.