How TrustZone could be bypassed: Side-Channel Attacks on a modern System-on-Chip

Sebanjila Kevin Bukasa[12], Ronan Lashermes[12], Hélène Le Bouder[23], Jean-Louis Lanet[12], and Axel Legay[2]

[1] LHS-PEC
[2] TAMIS INRIA {firstname.lastname}@inria.fr
[3] IMT Atlantique helene.lebouder@imt-atlantique.fr

Abstract. Side-channel attacks (SCA) exploit the reification of a computation through its physical dimensions (current consumption, EM emission, . . .). Focusing on Elecromagnetic analyses (EMA), such analyses have mostly been considered on low-end devices: smartcards and microcontrollers. In the wake of recent works, we propose to analyze the effects of a modern microarchitecture on the efficiency of EMA (here Correlation Power Analysis and template attacks). We show that despite the difficulty to synchronize the measurements, the speed of the targeted core and the activity of other cores on the same chip can still be accommodated. Finally, we confirm that enabling the secure mode of TrustZone (a hardware-assisted software countermeasure) has no effect whatsoever on the EMA efficiency. Therefore, critical applications in TrustZone are not more secure than in the normal world with respect to EMA, in accordance with the fact that it is not a countermeasure against physical attacks. For the best of our knowledge this is the first application of EMA against TrustZone.

Keywords: ARM TrustZone, Side-Channel Analysis (SCA), Raspberry Pi 2

1 Introduction

Embedded devices have never been so pervasive in our everyday life. Smartphones are already widespread and will continue to be present all around the world. Internet-of-Things (IoT) devices are poised to become ubiquitous. In parallel, more diverse and more critical applications are accessed with these devices: credit cards or other payment methods such as Apple Pay [5,23], password managers [10] or two factors authentication (2FA) are some examples.

Considering the information security, these devices offer a new set of possible threats: the attacker can have physical access to the device. Think of a stolen smartphone or a set-top box allowing access to pay-TV whose user would prefer the TV without having to pay.

These problems have already been faced by the smartcard industry in the past [16].

Yet apart from embedded secure elements (eSE, which are basically smartcard IPs in a chip), modern System-on-Chip (SoC) fail to implement countermeasures against physical attacks. The reasons may be the highly competitive market for these chips that do not play well with the prudent pace needed to obtain secure devices. The fact that the sensitivity of these modern SoCs to physical attacks is not well known does not support a much needed consideration for these attacks.

Modern chips are increasingly complex, including several peripherals, working at high clock speed, including several memory levels, numerous cores, and security extensions such as TrustZone (defined in section 2), the influence of all this activity on a SoC is of particular interest when it comes to physical attacks. This is what we want to verify in this paper by experimentally setup physical attacks.

Motivations: Our main objective is to expose the risks faced by users using embedded devices for critical operations when they are not protected against physical attacks. Most physical attacks are performed on low-end devices (microcontrollers and smartcards) and we investigate if they can be transposed easily to high-end devices. We verify that the TrustZone implementation does not offer any protection against side-channels leakages (TrustZone is only advertised as a software security solution by ARM, not hardware), and even multicore computations has no influence on these leakages.

Contribution: Two different attacks have been experimentally realized on a device representing a modern smartphone. We will use two use-cases for that purpose: the secret-key cryptography key recovery, and a Personal Identification Number (PIN) recovery.

These attacks have been realized while monitoring the impact of different features of the device.

Organization of this paper: The paper is organized as follows. Trusted Environment Execution and TrustZone specific implementation are presented in section 2. In section 3, Side-channel attacks (SCA) are presented and our two use cases are detailed. Our setup and experimental results are given in section 4. Finally, the conclusion is drawn in section 5.

2 Trusted Execution Environment and TrustZone

2.1 Trusted Execution Environment (TEE)

Trusted Execution Environment Protection Profile (TEE PP) is a profile specified by GlobalPlatform in 2014 [1]. This TEE PP has been made specifically for mobile devices, the goal was to deal with new threats in the mobile world and to allow customers to have confidence in the TEE in order to deliver mobile wallets, or secure mobility solutions.

A TEE is an execution environment that respect the TEE PP, meaning that it can be trusted by users. A Trusted Application (TA) is an application running in the TEE, it provides security related functions, and its assets include code, program data, cryptographic keys, etc. A TEE is designed to provide isolated execution, integrity of TA but also integrity and confidentiality of TA assets.

In order to provide a secure environment, a TEE might be working alongside a Rich Execution Environment (REE). A TEE is in fact another execution environment with its own Operating System (OS), the Trusted OS (TOS) and its own applications, TAs, executing over it. The OSes are isolated and the Rich OS (ROS) has just one entry point to the TOS.

There are several implementations of TEEs. The most notable are ARM TrustZone, detailed in this paper and Intel SGX [11].

2.2 TrustZone

TrustZone (TZ) is a specific implementation of a TEE proposed by ARM. This proposition is based on a few hardware-specific parts only. Security functions are mainly performed by software implementations or by optional pieces of hardware that can be added by each integrator. It respects GlobalPlatform specifications, and goes into TEE PP.

TrustZone hardware architecture leads to specific software implementations, a ROS, a TOS and the entry point (managing accesses to the secure world).

It consists in separating hardware resources into two worlds, Secure and Normal (Non-Secure) worlds. This is achieved by adding a security-related bit on the system bus (and therefore all bus accesses are impacted), and by adding new privilege levels. In that way each peripheral, each data or code section are separated between the two worlds and a core can only handle them if it is in the right privilege mode.

Typically, a non-secure ARMv7 core has three privilege modes:

- PL0 or user mode is the least privileged and is the mode where user application are executed in.
- PL1 or kernel mode is an intermediary privileged level and is typically where the kernel (of the ROS) is executed.
- PL2 or hypervisor mode is the highest level and is where the hypervisor (if any) is executed.

Now the TZ adds new modes. In addition to the previously defined non-secure PL0, PL1 and PL2, the new modes are:

- Secure PL0, the secure user mode is where trusted applications are executed.
- Secure PL1, the secure kernel mode is the mode for the secure kernel execution for the TOS.
- Monitor mode is a new mode that must be used to transition between the non-secure and the secure worlds.

When the core is in a secure state, the *non-secure (NS)* bit on the bus is set to 0. As a consequence, the systems on the bus can adapt to the security state (e.g. a peripheral may forbid the access when *NS* is 0).

3 Side Channel Attacks

3.1 Definition

When discussing about the security of an algorithm, numerous mathematical tools allow developers to prove its security. Unfortunately those tools cannot consider the interaction of the computing unit with its physical environment. Physical attacks are a real threat, even for algorithms proved secure mathematically. Side-channel analyses (SCA) are physical attacks based on the observation of the circuit behavior during a computation. They exploit the fact that some physical quantities depend on intermediary values of the computation in the device. This is the so-called leakage of information. The most classic leakages are timing [17,15], power consumption [21] and electromagnetic emissions (EM) [27]. In this paper we will focus on Electromagnetic analyses (EMA).

Side-channel analyses can be split in three families.

- The Simple Power Analysis (SPA) [20] uses directly the leakage to obtain the secret information.

- More complex attacks that need a mathematical model for the leakage and a statistic tool called a distinguisher. These attacks have generally a divide-and-conquer approach. The first ones were the Differential Power Analysis (DPA) [26] and Correlation Power Analysis (CPA) [7]. That is why this family of Side-Channel Attack (SCA) is often called the "DPA/CPA attacks". The differences between attacks of this family generally come from the choice of the model and the distinguisher.
- Template attacks are based on a statistical classification which replaces the mathematical model for the leakage in a DPA. To implement a template attack, a pair of identical devices is needed. One is called the profiling device, the attacker has full control of it, and is used to learn the leakage characteristics. The other is the targeted device on which the attack is carried. Template attacks were introduced as the strongest possible side-channel attacks from an information theoretic point of view [8]. They have been used in many attacks, such as [3,6] since.

SCA are threats for all standard cryptosystems as Data Encryption Standard (DES) [7,26], Advanced Encryption Standard (AES) [14,20], RSA cryptosystem [12], Elliptic Curve Cryptography (ECC) [25] and for critical applications not using cryptography, e.g. PIN verification [6]. SCA can also be used to reverse engineer algorithms [9].

3.2 Previous works

Historically, the targeted devices were mostly smartcards, microcontrollers or cryptographic coprocessors [16] since they were the only embedded devices with critical information on it (for credit cards and pay-TV notably). These devices are relatively slow (under 10MHz) and have simple architectures: a simple cache architecture if any, shallow pipeline, ... Features that make SCA easier. As a consequence, the smartcard industry developed strong countermeasures against physical attacks.

However, today the same applications are more and more executed on smartphones, and they do not build upon the knowledge from the smartcard industry in terms of SCA. The chips used in these devices are full featured processors running at high speed (1GHz+, > 100X faster than a smartcard), with complex architectures (multiple cache levels, deep pipelines). SCA are therefore more complex. Measurement equipments have to be adapted to accept much higher frequencies. Jitter (temporal variance) is more important due to the caches, making synchronization harder. Yet SCAs on smartphones have already been performed. In [2], the authors have developed an EMA on an Android phone (running at 400MHz). They targeted the secret key of an AES implementation in the Android BouncyCastle library. They show a key recovery in 250 traces with their special techniques designed to take into account the desynchronization added by the java virtual machine behavior.

How to do EMA on fast modern systems has been described in [4,19]. They both target a Beaglebone Black board running at 1GHz. They had to filter out traces that where not properly synchronized due to interrupts and other operating system operations. Resynchronization and filtering are necessary to obtain clean traces. In [4], the authors attacked their own optimized AES implementation and show that their countermeasures are effective. In [19], the authors compared the leakage of an AES on the ARM core with respect to the NEON coprocessor.

From these papers, we can see that transposing EMA on high end processors requires a lot more experimental work. In particular, synchronization can be a problem because of a virtual machine (in [2]) or because of the operating system (in [4,19]). That in why in our case, we work in a bare-metal environment (i.e. without an operating system) to focus on the microarchitecture influence on the efficiency of EMA.

A second category of EMA on smartphone has been performed targeting public-key cryptography (RSA [12,29] or ECC [13]). Synthetically, these attacks have the same leakage principle. Public-key cryptography deals with big numbers (3072-bit for RSA, 256-bit for ECC at the 128-bit security level). Handling these big numbers in software requires to split them in words whose size is determined by the chip architecture (32-bit or 64-bit in most architectures).

As an example, a multiplication of two big numbers requires looping over the words constituting the operands. The sequence of operation in each iteration has a leakage signature that can be identified in the frequency domain (since the iteration is repeated several times). Interestingly, the leakage in the frequency domain occurs at a relatively low frequency compared to the clock frequency (it corresponds to several instructions). Thereby, the attacker can identify the leakage signature of a big numbers multiplication versus squaring (for RSA) or point addition versus point doubling (for ECC). From this leakage, the authors devised methods to recover the secret.

Timing attacks are possible on high-end devices, in particular [18,30] highlighted that cache timing attacks can be performed in order to retrieve some informations on mobile devices, even from TrustZone.

3.3 Use cases

To test the possibility to attack a modern SoC with EMA, two representative use-cases have been chosen: a CPA on the AES and a template attack on a Verify PIN algorithm.

CPA on AES The first use-case is the well established CPA [7] applied to the AES.

Targeted algorithm The Advanced Encryption Standard is a standard established by the NIST [24] for symmetric key cryptography. It is a block-cipher, the encryption first consists in mapping the plaintext T of 128 bits into a two-dimensional array of $4 \cdot 4 = 16$ bytes called the State (4 rows and 4 columns). Then, after a preliminary xor (noted $\oplus$) between the input and the key K_0, the AES executes 10 times a round-function that operates on the State. The operations used during these rounds are:

- **SubBytes**, composed by non-linear transformations: 16 S-boxes noted SB, working independently on individual bytes of the State. In the targeted implementation, the S-boxes use a lookup table.
- **ShiftRows** noted SR, a byte-shifting operation on each row of the State.
- **MixColumns** noted MC, a linear matrix multiplication on $GF(2^8)$, working on each column of the State.
- **AddRoundKey** a xor between the State and the round-key K_r, $r \in [\![0, 10]\!]$.

The attack In this attack the target is the key of the first round K_0. To have a divide-and-conquer approach, K_0 is attacked byte per byte. There are only 256 possible values for each byte. The leakage is the EM emission measured at the S-boxes output. It depends on known plaintexts T and the secret key K_0.

So for each text T, and each guess k in $[\![0, 255]\!]$ a prediction P can be computed.

$$P(T, k) = SB(T \oplus k)$$

In particular it can be noted that the leakage is “value-based” and not, as usually the case, the Hamming Weight (HW) of the value (no leakage is observed in the HW model experimentally).

These predictions are confronted to the traces collected with a Pearson correlation that shows a linear relation between measurements and predictions.

Template attack on a Verify PIN algorithm The attack summarized here is the attack of Le Bouder *et al.* presented in [6].

The target: Verify PIN algorithm In many devices, a Personal Identification Number (PIN) is used to authenticate the user. The main protection in the use of a PIN comes from the fact that the user has a limited number of trials. A PIN is an array of m digits. A True PIN is embedded in the device; and a Candidate PIN is proposed by the user. The different values taken by the PIN are defined in $\llbracket 0, 9 \rrbracket^m$. The Verify PIN is the algorithm which tests if the Candidate PIN is correct or not. A good countermeasure against fault attacks is to compare the correct and Candidate PIN twice. The algorithm for the PIN comparison in constant time, from [28], is shown in algorithm 1.

Algorithm 1 Comparison between candidate PIN and true PIN

```
procedure COMPARISON(candidate PIN $CP$, true PIN $TP$)
    status = FALSE
    diff = FALSE
    fake = FALSE
    for $b = 0$ to $m - 1$ do
        if $CP[b] \neq TP[b]$ then
            diff = TRUE
        else
            fake = TRUE
        end if
        if ($b = m - 1$) and (diff = FALSE) then
            status = TRUE
        else
            fake = TRUE
        end if
    end for
    return status
end procedure
```

The attack The goal of this attack is to retrieve the true PIN even if the the number of trials is limited. That is why a template attack is used. It is supposed that an attacker has a profiling device and she can:

- change the true PIN in her profiling device;
- obtain many traces on her profiling device.

The attack starts with a measurement campaign on the profiling device. Template attacks use a divide-and-conquer approach, the digits of the true PIN are attacked separately. For each value of the pairs, candidate PIN and true PIN (one digit each), (v, k) in $\llbracket 0, 9 \rrbracket^2$, traces are collected. $M_{v,k} = \left\{ xk_{(i,j)} \right\}$, i for trace, j for time.

For effective computations, both for proper floating point arithmetic and to limit the amount of data to handle, a principal component (PCA) preprocessing is used on the data and the matrix $M_{v,k}$ is only considered here after this preprocessing step. After the PCA, our traces have always 10k points each.

The covariance matrix $S_{v,k} = \left\{ sk_{(j,j')} \right\}$ is computed.

$$sk_{(j,j')} = \frac{1}{n-1} \cdot \left(xk_j - \overline{xk}_j \right)^t \left(xk_{j'} - \overline{xk}_{j'} \right) \quad .$$

On the targeted device, a first trace $T_v = \{x_j\}$ is collected, where all the candidate PIN digits are equals to $v = 0$. For each digit the attacker confronts the trace T_v to the template matrix $S_{v,k}$, with the general formula in template attacks (Mahalanobis distance):

$$F_v \left(T_v | S_{v,k}, \overline{xk} \right) \propto \cdot \exp \left(-\frac{1}{2} \cdot \left(T_v - \overline{xk} \right) \cdot S_{v,k}^{-1} \cdot \left(T_v - \overline{xk} \right)^t \right) \quad .$$

The attack returns the guess k_v for which F_v is maximal for a given T_v, or ranks the guesses k according to the value of $F_v(T_v, k)$.

4 Experiments

4.1 Targeted device

A chip which is representative of what can be found in modern smartphones is targeted. Unfortunately, the public documentation on modern SoC is often sparse. To ease our bare-metal development process, we chose a relatively open platform with some documentation (mostly thanks to the community around it): the Raspberry Pi 2 (RPi2). This board possesses a BCM2836 SoC from Broadcom and features:

- a quad-core Cortex-A7 processor (ARMv7) running at 900MHz,
- a VideoCore IV GPU running at 250MHz,
- 1GB of LPDDR2 RAM memory running at 400MHz (or 450MHz in Turbo mode),
- TrustZone is available.

The processor uses 2 levels of cache:

L1 : 32KB, 4-way associative, separate data and instructions,

L2 : 512 KB, 8-way associative, unified data and instructions, shared between GPU and CPU.

The RPi2 board features GPIOs allowing to send electric signals directly from our code. In particular, a GPIO has been used to obtain a trigger when our targeted code is executed. This signal is used by the oscilloscope to know when to measure the leakage.

4.2 Software implementation

Since we want to measure the impact of the micro-architecture on the side-channel leakage, our targeted applications run on bare-metal (without OS). Therefore there is no context switching due to the processes scheduling or any other interruptions during the measured computation.

Our custom kernel is an aggregation of different functions, it contains two applications. First **AES** is a simple naive AES implementation without any countermeasures. Plaintexts are sent to the board through the UART and ciphertexts are returned. **PIN** is the other application, implementing a secure *verify pin* algorithm (from [28]). PIN candidates are sent to the board through the UART which answers if it is correct or not (for experimental purposes, an unlimited number of tries could be done without the secret destruction).

Both applications are always present in the final binary and one is selected at start-up with a proper command sent on the UART. Additionally, a configuration can be chosen at this step to enable or disable several features, namely the TrustZone activation, and the multicore execution. The following configurations have been tested:

1. *Default (D)*: the applications are run in non-secure kernel mode on a single core. The other cores are trapped in an empty infinite loop. Data and instruction caches are enabled but not the MMU.
2. *Multicore (M)*: same as default but now the other cores are active. They generate two pseudo random numbers (using the xorshift [22] algorithm) and compute their GCD in an infinite loop. This simulates an intense computation on all cores (asynchronously from our targeted applications).
3. *Secure (S)*: same as default but the application is executed in secure kernel mode. The other core are inactive.
4. *Secure and Multicore (S+M)*: this configuration executes our application in secure kernel mode while the other cores are active, computing GCD in non-secure kernel mode.

4.3 Test Bench

Our EM measurement test bench includes the following devices:

- An EM probe from Langer either a *RF-R 0.3-3* or a *RF-R 400-1* both with a 30MHz to 3GHz bandwidth (see below),
- a Langer PA 303 preamplifier (3GHz bandwidth) to amplify the signal from the probe,
- a DSOS404A oscilloscope from Keysight with a 4GHz bandwidth able to capture up to 20 GSamples per second.

A control computer (Xeon E5-1603v3 @ 2.80GHz, 4 cores, 40GB RAM) is used to orchestrate the measurements and perform the analysis with home-made tools. In particular, special care was needed when developing our tools to achieve a nice measurement speed, and be able to manage the vast quantity of data that resulted from the experiments. Around 7 hours and 100GB of free disk space are necessary to capture the needed traces for the template on our test bench, and 70 hours to perform an attack.

4.4 Preliminary experiments

Now that the applications and the test bench have been described, preliminary experiments need to be done in order to find information leakage. Where should the EM probe be located, what can we see at this location?

For this part, a *RF-R 0.3-3* probe from Langer was used (bandwidth from 30MHz to 3GHz). The probe is moved over the board (not only the chip since leakage can occur at other locations) and on both sides of the board.

Then data is measured for 250ns both during a real sbox computation and when no computation at all is performed (idle state).

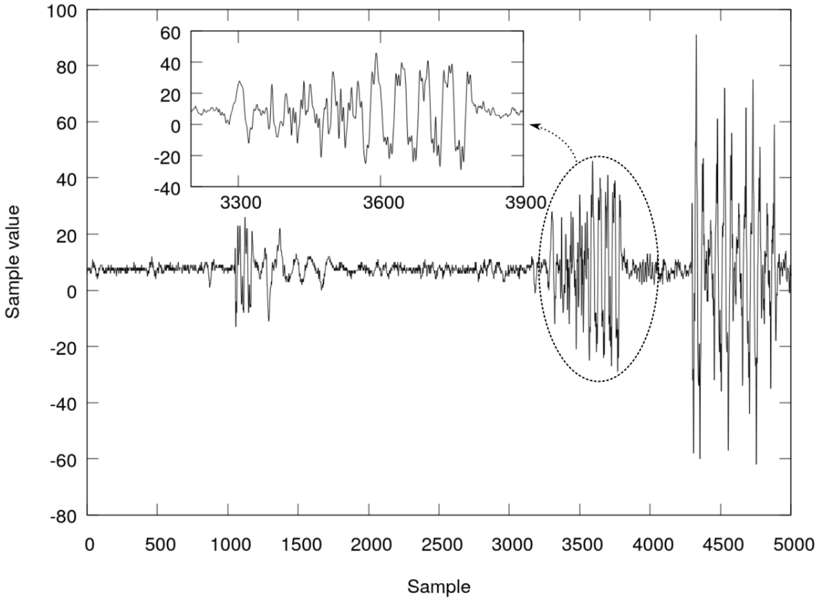

Fig. 1: A trace measured during an s-box computation. A pattern which may be a memory access is stressed out.

A measured trace is shown on figure 1 corresponding to an s-box computation. Patterns can clearly be seen but do not seem synchronized with our computation. A spectrum measurement shows that this pattern main frequency is 400MHz which corresponds to the RAM memory frequency.

Yet even in the presence of these patterns, no leakage was detected with the small *RF-R 0.3-3* probe.

Using the big *RF-R 400-1* probe instead, the same patterns are present but a leakage can now be detected.

Finally, after a trial-and-error phase, we settle to use the big *RF-R 400-1* probe on the back side of the board since this configuration was the best for us to detect a leakage with a CPA. We cannot guarantee that there is no location were the leakage is more spatially concentrated in the absence of a systematic spatial characterization of the leakage (requiring additional apparatus).

4.5 Experimental results

CPA The first attacks are CPA on the first round of an AES encryption. Since we are mostly interested in the leakage, we focus on the recovery of one key byte only, yet the attack can easily be generalized to retrieve the full key. The S-boxes are implemented using a look-up table (C array) and no countermeasures are used.

What does not work To perform the attack, various tries (probe, synchronization points, models) have to be done to find a leakage on the *default (D)* configuration.

We do not find any leakage with the small *RF-R 0.3-3* probe, whatever the location (which does not mean that a leaking location for this probe does not exists). The *RF-R 400-1* probe is used instead and allows to find a leakage.

The classic Hamming Weight model (meaning that the EM leakage is linearly dependant on the Hamming Weight of the value of interest) does not work. We find the EM leakage is directly linked to the value instead. This was not expected and may hint that new leakage models must be developed for modern SoC.

Finally, the synchronization is the most difficult part of an experimental SCA. In our case, our application runs on bare-metal and no synchronization issue can be caused by an operating system or a virtual machine. Nonetheless, S-boxes are implemented with look-up tables. Meaning that desynchronization can be caused by the memory hierarchy fetching the s-box result. Indeed, when measuring the duration of an s-box with a GPIO raised and fallen before and after the s-box, we observe an important jitter (which can also be attributed to the GPIO subsystem). In order to properly synchronize trigger signal on the first s-box result, a GPIO is raised just before the second s-box execution. Since this second execution is done as soon as the first result is obtained we are synchronous with the first s-box result.

CPA leakage analysis The CPA results can give information on the leakage characteristics. A CPA is carried on 200k traces of 5k points under 20 minutes: 18 minutes to acquire the data and 2 minutes to compute the CPA results.

The highest positive correlation peak lasts around 20 samples, i.e. 1ns. By comparison, the core clock has a 900MHz frequency, i.e. a clock cycle lasts 1.1ns. In particular, it means that our measurement jitter is low enough to detect a leakage that lasts $1ns$. Additionally it explains why synchronization is so hard on high end devices: a 2ns jitter can add a significant noise over a 1ns signal (by comparison, light can travel only around 30cm in 1ns). Smaller correlation

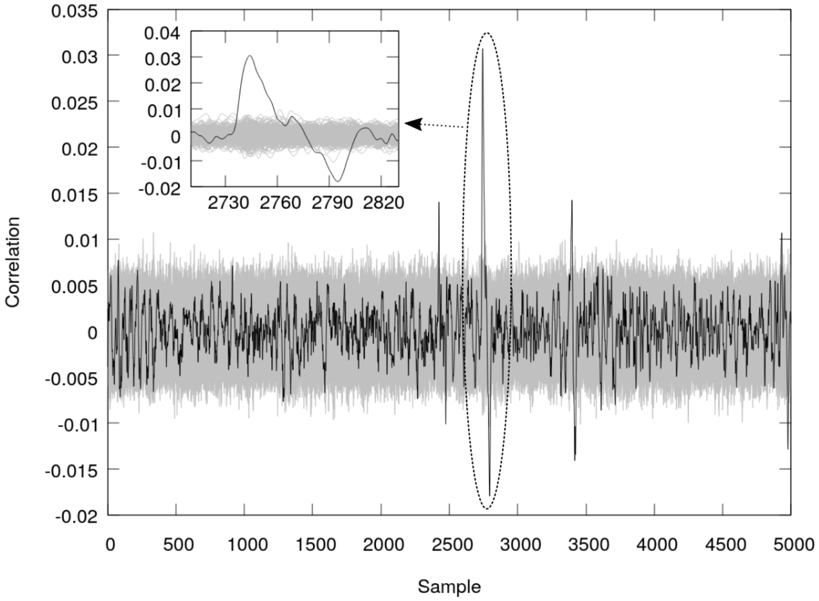

Fig. 2: CPA result for the Default (D) configuration, 200k traces and 20GS/s (smallest features take 7 samples as per the maximum bandwidth (3GHz)). Black is for the good key value, grey for all others.

peaks can be observed: one is 12ns before the main peak, a second is 41ns after the same main peak (timing measured for positive peak).

All positive peaks are followed by a negative one, this common feature in CPA is usually explained by a value-dependent temporal shift, i.e. in the Hamming Weight model, a 1 or a 0 on a wire have different associated capacitance and as a consequence different speed. However in our settings, the Hamming Weight model is not correct and therefore this explanation does not hold. Yet we did not found a proper explanation for this phenomenon.

We cannot have any certitude to explain these peaks without further information on the chip microarchitecture. Yet the timings suggest a fast memory access (cache): we may be visualizing the data transiting in the memory hierarchy.

CPA results for the different configurations In this paragraph, the influence of the different configurations is tested on the CPA results.

The correlation values as shown on table 1 around 3%, is low in all cases. The leakage signal is hidden behind the noise generated by the numerous SoC subsystems. Yet the power of the CPA means that the secret can be found even in noisy cases. The results shown here required $200k$ traces acquired in 18 minutes.

The leakage peaks appeared nearly at the same sample for all configurations, simply meaning that the S and M configurations do not impact the timing of the leakage (no cache/memory interference).

Table 1: For 5000 samples, 200k traces and a total duration of 250ns (20GS/s)

Configuration	Maximum correlation	Peak sample value
`D`	3.1%	2745
`S`	3.4%	2746
`M`	3.7%	2746
`S+M`	3.6%	2747

Template attacks The second attack is a template attack on a Verify PIN algorithm. At first, the small *RF-R 0.3-3* probe has been used. Attacks where successful but results were not reproducible: any small variation in our setup (probe position in particular) gave different results.

Since we are interested in the variation of the efficiency of the attack with or without some microarchitecture features enabled, the big *RF-R 400-1* probe was preferred (giving reproducible results).

Finally, we use the same *RF-R 400-1* probe for both attacks.

The results of these attacks are presented in the table 2. One attack (one row of table 2) requires 7 hours of measurements and 70 hours of computation split as follows:

- 1 hours to perform the PCA learning step (create the transformation matrix),
- 3 hours to apply the PCA transformation over all acquired traces,
- 10 hours to compute the template classes,
- 50-60 hours to perform the attack on the 150k test traces.

The template attack [6] is particularly efficient to detect whether the secret PIN digit is equal to the candidate or not. Here, the success metric used is the ratio of successful detection that the secret PIN digit is equal to the candidate (for 10 digit values, a purely random method would have a 10% success rate).

For the 10 secret digit values (for 1 candidate value only), template classes where created from the measurement of 200k traces. In other words, for each secret value, 200k traces where measured and used to characterize the signal corresponding to this secret (forming one template class). This is the learning dataset.

Then 15k additional traces where measured, here again for each secret value. This is the test dataset. Our success rate is measured by trying to match these $10 \cdot 15k = 150k$ traces to each one of the 10 template classes.

Let r be the actual, true, success rate and p the measured success rate. We can define the maximum error as a value E guaranteeing $|p - r| < E$. Let σ be our confidence level, i.e. $\sigma = 3$ means that we have a 3 standard deviations confidence ($\approx 99.7\%$).

We can link our maximum error and our confidence in this error with the equation 1.

$$E = \sigma \cdot \sqrt{\frac{p \cdot (1-p)}{n}} \tag{1}$$

where n is the number of measurements (normal approximation method of the binomial confidence interval).

For $p = 0.5, n = 15000$ and $\sigma = 3$, we obtain $E = 1.22\%$ (E is maximized for $p = 0.5$). In the following result tables, the success rate is given with a confidence of 3 standard deviations (99.7%) that the error is below 1.22%.

Table 2: Success rates of the template attacks.

Setup	Sampling rate	Measure duration	Success rate
`D`	10 Gsamples/s	$2\mu s$ (20kpts)	37.84%
`S`	10 Gsamples/s	$2\mu s$ (20kpts)	38.3%
`M`	10 Gsamples/s	$2\mu s$ (20kpts)	37.88%
`S+M`	10 Gsamples/s	$2\mu s$ (20kpts)	36.40%
`D`	20 Gsamples/s	$2.5\mu s$ (50kpts)	35.19%
`S`	20 Gsamples/s	$2.5\mu s$ (50kpts)	32.85%
`M`	20 Gsamples/s	$2.5\mu s$ (50kpts)	18.45%
`S+M`	20 Gsamples/s	$2.5\mu s$ (50kpts)	17.81%

First, we observe on table 2 that with a 10GS/s sampling rates, the attacks success rate is approximately 37.3% independently of the configuration (all results are in our error interval of 1.22%). **The TrustZone and the activity of other cores have no effect on the attack success rate.**

It can be observed on the last two rows of table 2 that using a bigger sampling rate can **decrease** the success rate, below our error interval. We attribute this paradoxical effect to our principal component analysis (PCA) preprocessing that select the dimensions maximizing the variance. Since at the end of the PCA 10kpts traces are obtained whatever the size of the input traces (20kpts or 50kpts), it is possible that the preprocessing reduces our success rate by maximizing variance not linked with the secret, such as variance generated by the activity of other cores.

Therefore, the preprocessing is able to efficiently compress the template data (traces size divided by 5), but at the cost of a reduced success rate. Template attacks may be less sensitive to bad synchronization but they require much bigger computing resources and one has to be careful with the preprocessing (as table 2 shows).

5 Conclusion

The difficulty of adapting side-channel attacks to high-end devices has often been supposed. We have seen that the true difficulty arises from the need for a more precise synchronization with the leakage.

If the attacker is able to synchronize, simple attacks (CPA) have been shown to be successful. The activity of other cores and the utilization of TrustZone have no effect on the attacks efficiency. As expected the TrustZone, a hardware-assisted software protection mechanism, offers no SCA protection. It may be a problem when most TrustZone capable devices are embedded systems that may endure physical attacks.

Yet if synchronization is not possible, there are some attacks (e.g. template attacks) that are able to exploit the time dimension as any other dimension. We hint that such techniques may be more common in the future to overcome the true difficulty with high-end devices: dealing with time precision (problem even worse with an OS or a virtual machine). Here again TrustZone or the activity of other cores have no incidence. But with these attacks, managing the big amount of data generated by our measures may prove to be the limiting factor, requiring better computing resources.

Even if side-channel attacks techniques have to be adapted to the new challenges created by high-end devices, there is no reasons that they will not be as successful as on smartcards. The smartphone industry in particular should rapidly move to SCA-proof technologies before the development of these attacks.

References

1. TEE Protection Profile, http://www.commoncriteriaportal.org/files/ppfiles/anssi-profil_PP-2014_01.pdf
2. Aboulkassimi, D., Agoyan, M., Freund, L., Fournier, J., Robisson, B., Tria, A.: Electromagnetic analysis (ema) of software aes on java mobile phones. In: 2011 IEEE International Workshop on Information Forensics and Security. pp. 1–6 (Nov 2011)
3. Archambeau, C., Eric Peeters, Standaert, F.X., Quisquater, J.J.: Template attacks in principal subspaces. In: Cryptographic Hardware and Embedded Systems-CHES 2006, pp. 1–14. Springer
4. Balasch, J., Gierlichs, B., Reparaz, O., Verbauwhede, I.: DPA, Bitslicing and Masking at 1 GHz, pp. 599–619. Springer Berlin Heidelberg, Berlin, Heidelberg (2015), http://dx.doi.org/10.1007/978-3-662-48324-4_30
5. Betters, E.: Apple pay: How it works (2016), http://www.pocket-lint.com/news/130870-apple-pay-explained-what-is-it-and-how-does-it-work, accessed: 2017-02-14
6. Bouder, H.L., Barry, T., Couroussé, D., Lashermes, R., Lanet, J.L.: A template attack against VERIFY PIN algorithms. In: Secrypt (2016)
7. Brier, E., Clavier, C., Olivier, F.: Correlation Power Analysis with a Leakage Model. In: CHES. pp. 16–29 (2004)
8. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: Cryptographic Hardware and Embedded Systems-CHES 2002, pp. 13–28. Springer (2003)
9. Clavier, C.: An improved SCARE cryptanalysis against a secret A3/A8 GSM algorithm. Information Systems Security pp. 143–155 (2007)
10. Corpuz, J.: Mobile password managers (2017), http://www.tomsguide.com/us/pictures-story/662-best-mobile-password-managers.html, accessed: 2017-02-14
11. Costan, V., Devadas, S.: Intel sgx explained. Cryptology ePrint Archive, Report 2016/086 (2016), http://eprint.iacr.org/2016/086
12. Genkin, D., Pachmanov, L., Pipman, I., Tromer, E.: Stealing keys from pcs using a radio: Cheap electromagnetic attacks on windowed exponentiation. In: International Workshop on Cryptographic Hardware and Embedded Systems. pp. 207–228. Springer (2015)
13. Genkin, D., Pachmanov, L., Pipman, I., Tromer, E., Yarom, Y.: Ecdsa key extraction from mobile devices via nonintrusive physical side channels. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 1626–1638. CCS '16, ACM, New York, NY, USA (2016), http://doi.acm.org/10.1145/2976749.2978353

14. Gierlichs, B., Batina, L., Tuyls, P., Preneel, B.: Mutual information analysis. In: International Workshop on Cryptographic Hardware and Embedded Systems. pp. 426–442. Springer (2008)
15. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Advances in Cryptology—CRYPTO'96. pp. 104–113. Springer (1996)
16. Koeune, F., Standaert, F.X.: A Tutorial on Physical Security and Side-Channel Attacks, pp. 78–108. Springer Berlin Heidelberg, Berlin, Heidelberg (2005), http://dx.doi.org/10.1007/11554578_3
17. Foo Kune, D., Kim, Y.: Timing attacks on pin input devices. In: Proceedings of the 17th ACM conference on Computer and communications security. pp. 678–680. ACM (2010)
18. Lipp, M., Gruss, D., Spreitzer, R., Maurice, C., Mangard, S.: Armageddon: Cache attacks on mobile devices. In: USENIX Security Symposium. pp. 549–564 (2016)
19. Longo, J., De Mulder, E., Page, D., Tunstall, M.: SoC It to EM: Electro-Magnetic Side-Channel Attacks on a Complex System-on-Chip, pp. 620–640. Springer Berlin Heidelberg, Berlin, Heidelberg (2015), http://dx.doi.org/10.1007/978-3-662-48324-4_31
20. Mangard, S.: A simple power-analysis (spa) attack on implementations of the aes key expansion. In: International Conference on Information Security and Cryptology. pp. 343–358. Springer (2002)
21. Mangard, S., Oswald, E., Popp, T.: Power analysis attacks: Revealing the secrets of smart cards, vol. 31. Springer Science & Business Media (2008)
22. Marsaglia, G., et al.: Xorshift rngs. Journal of Statistical Software 8(14), 1–6 (2003)
23. Nguyen, L.: Samsung pay: How it works (2016), http://www.androidauthority.com/samsung-pay-everything-you-need-to-know-678123/, accessed: 2017-02-14
24. NIST: Specification for the Advanced Encryption Standard. FIPS PUB 197 197 (November 2001)
25. Oswald, E.: Enhancing simple power-analysis attacks on elliptic curve cryptosystems. In: International Workshop on Cryptographic Hardware and Embedded Systems. pp. 82–97. Springer (2002)
26. Paul C. Kocher and Joshua Jaffe and Benjamin Jun: Differential Power Analysis. In: CRYPTO. pp. 388–397 (1999)
27. Quisquater, J.J., Samyde, D.: Electromagnetic analysis (EMA): Measures and counter-measures for smart cards. In: Smart Card Programming and Security, pp. 200–210. Springer (2001)
28. Riviere, L.: Sécurité des implémentations logicielles face aux attaques par injection de faute sur systemes embarqués. Ph.D. thesis, Telecom Paris Tech (2015)
29. Uno, H., Endo, S., i. Hayashi, Y., Homma, N., Aoki, T.: Chosen-message electromagnetic analysis against cryptographic software on embedded os. In: 2014 International Symposium on Electromagnetic Compatibility, Tokyo. pp. 314–317 (May 2014)
30. Zhang, N., Sun, K., Shands, D., Lou, W., Hou, Y.T.: Truspy: Cache side-channel information leakage from the secure world on arm devices.