Comparison of side-channel leakage on Rich and Trusted Execution Environments

Paul Leignac, Olivier Potin, Jean-Baptiste Rigaud, Jean-Max Dutertre, Simon Pontie

Authors' version
https://dx.doi.org/10.1145/3304080.3304084
Proceedings of the Sixth Workshop on Cryptography and Security in Computing Systems CS2 2019

Comparison of side-channel leakage on Rich and Trusted Execution Environments

Paul Leignac, Olivier Potin, Jean-Baptiste Rigaud, Jean-Max Dutertre[1] and Simon Pontié[2]

[1]Mines Saint-Etienne, CEA-Tech, Centre CMP, Departement SAS, Gardanne, F-13541, FRANCE
[2]CEA Tech, Centre CMP, Equipe Commune CEA Tech - Mines Saint-Etienne, Gardanne, F-13541, FRANCE
[1]firstname.name@emse.fr
[2]simon.pontie@cea.fr

Abstract

A Trusted Execution Environment (TEE) is a software solution made to improve security inside system on chip (SoC) based on ARM architecture. It offers a compromise between the functionality of the Rich Operating System (Rich OS), for example Android, and the security of a Secure Element (SE). ARM TrustZone separates the SoC between two worlds (Normal World and Secure World). The Trusted OS (the OS on the TEE) has several security mechanisms that isolate and secure its execution and data from the Rich OS and save it from data theft. If these mechanisms are made to prevent software attack from Rich OS, this paper proposes to take a look at the identification of data leakage from a TEE facing physical attack. In particular, how a side-channel analysis on electromagnetic (EM) emissions using the Test Vector Leakage Assessment (TVLA) methodology permits to identify the leakage and a correlation electromagnetic analysis (CEMA) can exploit the results.

1 Introduction

A Side-Channel Analysis (SCA) is an attack that exploits a leakage in the implementation of an algorithm by extracting information from a measured physical value which has 'correlated' to the target computation. It could be timing information [1], power consumption [2], electromagnetic emanations [3], or any other physical value. However, the evolution of SoCs and smartphones tends to move the attention to these targets that contain sensitive data (banking information, work content, personal data, etc.) and to develop new security technologies such as ARM TrustZone [4] and TEE [5, 6]. Some 64-bit microcontrollers (for example ARM Cortex-M35P) also combine software protection with TrustZone technology and additional physical protections. But mobile devices require advanced microprocessors, which at present, do not include side-channel attack protections. In this paper, we study how is the complexity of side-channel key extraction of AES encryption [7] when it is executed on Rich Execution Environment (REE) and Trusted Execution Environment. Particularly, we analyse whether the complexity of SoC used in mobile devices and the OS executed on — Rich OS or Trusted OS — can be an intrinsic protection.

1.1 Previous attacks on SoC

Attacks were performed in the last few years on advanced system. In [8], Aboulkassimi et al. attacked a software implementation of AES under JAVA ME using EM measures. In [9], Longo et al. added an OS and attacked a SoC running under Debian to extract AES encryption key exploiting electro-

magnetic emanations. Some attacks appeared on ARM TrustZone and TEE. It was mostly software attacks. In [10], Rosenberg used a security breach in the QSEE (Qualcomm TEE) to unlock the bootloader on a Motorola smartphone. In [11], Laginimaineb redirected some functions inside QSEE to be able to execute arbitrary code inside the TEE. Physical attacks were made on a TEE-based system. In [12], Tang et al. made a fault attack to modify the behavior of the clock and voltage regulators, violate the timing constraint and extract AES secret key on both Android and TEE. In [13], Zhang et al. extracted information by probing the cache during a software AES encryption on a TEE. In [14], Kevin et al. did side-channel EM analysis to extract cryptographic secret key in TEE. However, this was done on bare-metal (without OS).

We propose in this paper to manage a complete EM attack of an AES encryption on both REE and TEE OS. We discuss about what are the difficulties of the attack in terms of spatial and timing localisations of the AES encryption on complex SoC. What is the impact of TEE on a classical side-channel attack ?

1.2 Difficulties

The main difficulty for the attacker caused by the operating systems is that a lot of processes are running at the same time on the device. Also the presence of several cores as on all recent SoC may induce probing errors because of the scheduling of the execution flow between cores. The consequence is a lot of noise and interruptions which means it is more difficult to clearly identify the targeted execution, and have synchronized measures. To bypass these difficulties, we:

- Reduced the execution to one core.
- Identified the leakage with the TVLA methodology [15].
- Resynchronized the traces with mathematical tools.
- Extracted the secret data by electromagnetic correlation.

1.3 Targeted Device

The targeted board is an evaluation board LeMaker HiKey. The device component is a 64-bit SoC in 28nm technology. It has an octa-core ARM Cortex-A53 CPU with 5 available clock frequencies (208MHz to 1.2GHz). It is running Android Oreo (Android version 8.0) and Trustonic Secured Platforms for TEE (Kinibi). It is embedded in several smartphones on the market such as Huawei P8 Lite or Honor 5A.

1.4 Targeted Software

The software targeted on the the board is separated in two parts: the Client Application (CA), running under Android Oreo, and the Trusted Application (TA), running under the Trusted OS. Our application is a 128-bit AES encryption developed in C without countermeasure. The same key is fixed in both environment. In case of REE, the software executes an AES encryption in the Normal World. In case of TEE, the software execution goes through a very specific process and defined by Global Platform APIs [16] to produce the AES encryption in Secure World. Indeed, the CA starts from the Normal World user mode then goes to kernel mode and opens a session with the TA. Here, the chip switches to Secure World. A Monitor Mode is applied to save and restore OS context when switching between Normal and Secure Worlds. In Secure World, key, plaintexts buffer and memory used for the AES computing are stored in secure enclaves only reachable from TEE. Then, the TA executes the AES encryption. The same path backward permits to return to user mode in the Normal World. The CA regains control, closes the session with the TA and returns the cipher texts.

1.5 Non-intrusive attack setup

The targeted execution is an application performing a 128-bit AES encryption either on the REE or on the TEE. As same as [17], we measure the electromagnetic field around the chip through the package during the encryption aiming to do a side-channel analysis. Fortunately, the SoC is not a package-on-package with RAM on top which would imply to remove the RAM and potentially destroy the chip. To do so, we used a Teledyne Lecroy Wa-

veRunner 640Zi oscilloscope sampling at 10GS/s and with a 4GHz bandwith, a Langer H-field probe RF-B 0.3-3 with a 30MHz to 3GHz bandwidth and a Langer PA 306 amplifier with a 100kHz to 6GHz bandwidth.

As the chip contains 8 CPUs, we took X-Ray pictures of the SoC to identify 8-identical patterns and conclude the CPUs position. As the TA is executed in Secure World on the same CPU from which the CA calls the TA, we manage to target one of the CPUs and deactivate all others with CPU hotplug capability of the Linux kernel. On the assumption that any user with a root access to the Android side can modify the content of configuration files in the system, we are able to deactivate 7 cores. Moreover, we also manage to deactivated Dynamic Voltage and Frequency Scaling (DVFS) by setting the CPU frequency to the highest supported value. So, it permits to avoid dynamic time execution according to the CPU load. On an Android system, you can do it by configuring the CPU frequency scaling of the Linux Kernel.

2 Side-channel Attack

To prepare to the side-channel attack, we proceed to a reverse engineering analysis of the board on REE side to identify the leakage model of the circuit which permits to make assumptions on key values. We apply the Test Vector Leakage Assessment (TVLA) methodology. The methodology is described in [15] and consists in almost 1800 Welch's t-tests [18] knowing the plaintexts and the secret key. The results of specific t-tests regarding byte values gave us exploitable results which means that the leakage is linked to the value of the manipulated bytes. Consequently, we target a leakage model on Hamming weight on bytes for the side-channel attack. With this assumption , we can manage to do a Correlation ElectroMagnetic Analysis (CEMA) like the Correlation Power Analysis (CPA) described by Brier et al. in [19] except that we consider the EM traces instead of power traces. However, as we observe a lack of synchronization between traces during the TVLA process, we proceed to a pretreatment step to resynchronize 10,000 traces according to the process described in figure 1 before proceeding to CEMA. The attack was done on REE side to test the vulnerability of the hardware and on the TEE side to attack the trustzone.

2.1 Pretreatment Step

The resynchronisation is a two-step process involving the identification of a golden pattern and the pattern matching in the set of traces. The pattern matching steps are executed on each trace:

1. Compute cross-correlation between traces and golden pattern to select lower and/or upper time limits before fine grain pattern matching.
2. Decimate trace to decrease the trace size.
3. Find the best pattern in the trace which minimize the euclidean distance from the golden pattern. This search is bounded by time limits.
4. Estimate the quality of the pattern matching.

A part of an arbitrary chosen trace will be the golden pattern. Having analysed the chosen trace, the time frame that should be an AES encryption was identified. To avoid cache eviction or miss cache consequences, we fill the plaintexts buffer with a same plaintext. Consequently, if the first encryption causes cache misses, next encryptions must tend to a constant execution time. For example, we choose the third AES encryption among the ten successive iterations of the same AES encryption in figure 2. This part of the trace is saved and considered as a golden pattern. The next resynchronisation step consists in browsing all the traces to look for a pattern matching the golden pattern.

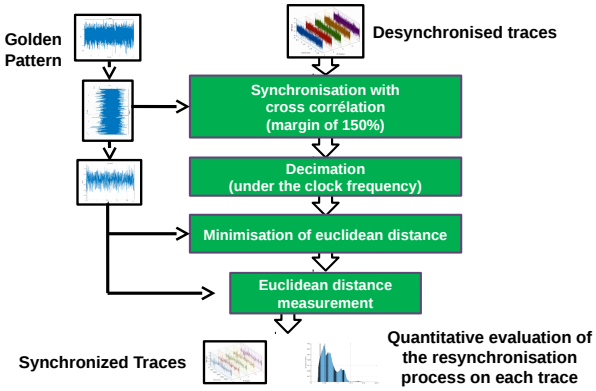

Figure 1: Traces resynchronisation process

For that, we compute the cross-correlation (1) coefficient between the golden pattern (X) and each trace (Y). The formula of the cross-correlation coefficient is given in Eq. 1:

$$\rho_{X,Y}(\tau) = \frac{1}{\sigma_X \sigma_Y} \gamma_{X,Y}(\tau) \quad (1)$$

Where:
$\rho_{X,Y}(\tau)$ = Cross-correlation between X and Y for a delay τ
σ_X = the standard deviation of X
σ_Y = the standard deviation of Y
$\gamma_{X,Y}(\tau)$=the cross-covariance between X and Y for a delay τ

The cross correlation coefficient is a measure of similarity between two time series. By applying it to the golden pattern and each trace, we measure if the trace matches the golden pattern. This operation is fast due to an optimisation based on fast Fourier transform. The result is only used to defined time limits around identified pattern. Step (2) allows us to decrease the number of samples in traces by decimation before proceeding to the next step. In the step (3), we find the delay τ that minimizes the euclidean distance between the golden pattern and a selected part of each trace around the time limits from step (1) (see Eq. 2). Once we have the delay τ_{delay}, we select in the trace a window of the same size of the golden pattern and starting at the instant $t = \tau_{delay}$. Step (1) and (2) reduce computation time of step (3). The result of step (4) indicates the quality of the golden pattern extraction in each trace which permits us to filter traces with low matching.

$$\tau_{delay} = \underset{\tau}{argmax}(\|X(t-\tau) - Y(t)\|) \quad (2)$$

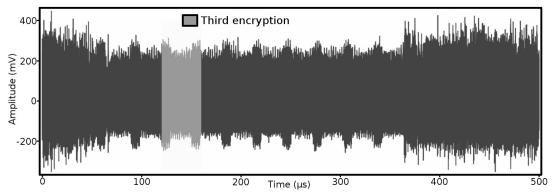

Figure 2: Trace of CA execution containing 10 AES

2.2 Experimental result on REE side

After the resynchronisation of the 10,000 traces in REE, we can proceed to the CEMA to measure the vulnerability of the chip in Normal World. To do the CEMA, we make 256-key hypotheses by computing the hamming weight of the 8-bit value of each of the 16 bytes of the AES state after the SubBytes transformation in the first round of the AES Encryption for all 10,000 plaintexts. Then, we compute the Pearson correlation coefficients between all the key hypothesis and the samples of all traces measured with the EM probe. For each byte of the key, the good candidate is the one whose correlation trace has the highest peak in absolute value and the instant in time this peak appears is the instant of the leakage. On the REE side, we were able to recover all the 16 bytes of the AES secret key with 10,000 traces in 30 minutes on a machine using 10 CPUs and with 32 GB of RAM (see Figure 3). The order of the found bytes for the key is: 0, 5, 10, 15, 4, 9, 14, 3, 8, 13, 2, 7, 12, 1, 6, 11. This corresponds to the order of the bytes after the ShiftRows transformation of the first round.

2.3 Experimental result on TEE side

The trustzone is exclusively a software countermeasure and, obviously, we can expect that the same algorithm executed on both environment REE and TEE has a same EM signature. We hope that the strategy developed on REE side permits us to reuse the golden pattern on the TEE side. Unfortunately, it was not! The measured EM field is much more noisy than in Normal World and the AES en-

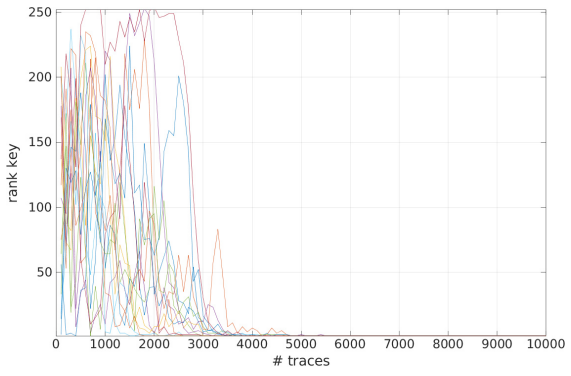

Figure 3: CPA the rank of the 16 correct keys on REE

cryptions were hard to distinguish. Moreover, the timing variability of the TA execution induced by the switch context between Normal World and Secure World increases the difficulty of resynchronisation. Another difficulty is the lack of GPIO in Secure World which allows us to trig on AES encryptions. Indeed, the access to GPIO in Secure World could be possible only if a kernel driver is available in TEE. Nevertheless, by running several times the TA with a different number of encryptions, we were able to identify an window inside the EM emanations whose length changed. This window length is 500 μs out of 12ms (the trigger width corresponding to the duration of the TA execution). But, with a time-frequency representation of the trace (see Figure 4), it becomes possible to extract a time frame including the 10 iterations of AES encryption.

A more precise time-frequency analysis permits us to clearly identify 10 patterns which correspond to the 10 rounds of the AES encryption. Consequently, we define these 10 patterns as the golden pattern for TEE encryption and apply the same strategy than in REE side. Finally, we were able to conclude the attack on the Trustzone and extract the secret key (see figure 5).

The bytes of the secret key have been found in this order: 15, 0, 5, 10, 4, 9, 14, 3, 8, 13, 2, 7, 12, 1, 6, 11. It corresponds to the order of the bytes in the state after the ShiftRows except for the first column that has been mixed, which could mean that the leakage appears during the MixColumns step. All key bytes are found with 6,000 traces (same order of magnitude as on the REE) in 45 minutes with the same resources allocation as for the REE attack.

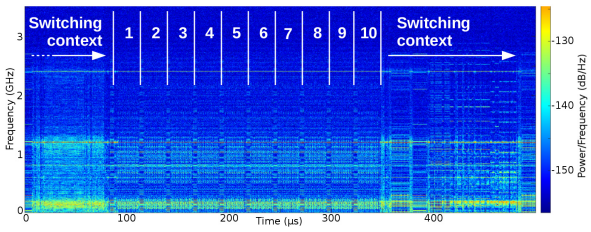

Figure 4: Spectrogram of the trace of TA execution containing 10 AES encryptions

3 Conclusion

We succeeded in performing the side channel analysis on both Android and Trusted OS. We proceeded in three steps: the identification of the leakage model with the TVLA method, an AES attack on REE side to test the hardware vulnerability of the SoC and an AES attack on TEE side. Most of the difficulties and strategy to bypass them were concentrated on the spatial and the timing localisation of correct AES encryptions: Identification of the 8-CPU cores on the SoC, set the execution on one CPU core, avoid as soon as possible the time shifting of the AES encryption process and filter EM traces with OS interruptions. Nevertheless, the SoC complexity and the TEE protection scheme were not sufficient against physical attack. However we were surprised that a same code on both side did not produce similar traces. It is still an open question why some discrepancies are observed between the REE and the TEE leakage and what are the causes of leakage sources ? In case of all the 8 CPU cores are activated, the attack should come to a successfull conclusion. The side-channel analysis would require more traces and time and another step of pretreatment to identify whether or not the trace is exploitable. It can be done by looking at the result of cross-correlation during pattern recognition process.

3.1 Perspectives and countermeasures

An attack on an AES T-Table implementation should still succeed but is it possible to proceed to

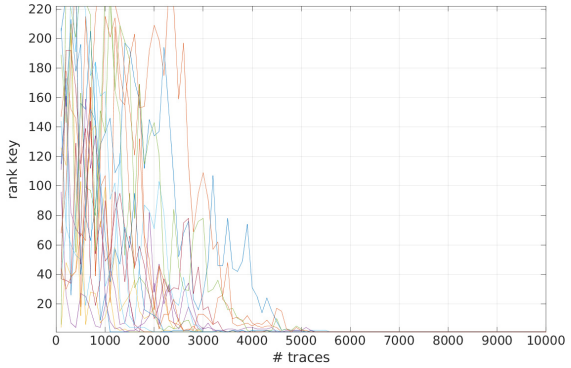

Figure 5: CPA rank of the 16 correct keys on TEE

the attack in case of more complex AES encryption algorithm (NEON based bitslicing) or dedicated assembler instructions of ARMv8 which implies to find few instructions in the complete TA process ? Of course, AES classical countermeasures (shifting or masking) should deal with the attack. As we observe the TA is vulnerable when the attacker can identify AES rounds, another proposition of countermeasure on the TEE side could be to interleave the rounds of the AES encryption with rounds of another AES that would be using random key and plaintext and these rounds would be randomly ordered. As there is no access to GPIO pins, the attacker has to use a cross-correlation to retrieve the AES in the trace. However, if there are some "false" rounds, the correlation would be fooled by the countermeasure and the analysis made on rounds from different AES encryptions, preventing the attack to success.

Acknowledgement

This work was carried out in the framework of the FUI-AAP20 project TEEVA supported by BPI France.

References

[1] P. C. Kocher, "Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems," in *Annual International Cryptology Conference*, pp. 104–113, Springer, 1996.

[2] P. Kocher *et al.*, "Differential power analysis," in *Advances in Cryptology — CRYPTO' 99* (M. Wiener, ed.), pp. 388–397, Springer Berlin Heidelberg, 1999.

[3] K. Gandolfi *et al.*, "Electromagnetic analysis: Concrete results," in *CHES 2001*, pp. 251–261, Springer, 2001.

[4] ARM Ltd, *ARM Security Technology - Building a Secure System using TrustZone Technology*, 2009.

[5] M. Sabt *et al.*, "Trusted execution environment: what it is, and what it is not," in *14th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, 2015.

[6] J.-E. Ekberg *et al.*, "Trusted execution environments on mobile devices," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pp. 1497–1498, ACM, 2013.

[7] P. FIPS, "197, advanced encryption standard (aes), national institute of standards and technology, us department of commerce, november 2001," 2001.

[8] D. Aboulkassimi *et al.*, "ElectroMagnetic Analysis (EMA) of Software AES on Java Mobile Phones," in *IEEE Intl. Workshop on Information Forensics and Security - WIFS'11*, (Foz do Iguaçu, Brazil), p. Paper 75, Nov. 2011.

[9] J. Longo *et al.*, "Soc it to em: Electromagnetic side-channel attacks on a complex system-on-chip," in *CHES 2015* (T. Güneysu and H. Handschuh, eds.), (Berlin, Heidelberg), pp. 620–640, Springer Berlin Heidelberg, 2015.

[10] D. Rosenberg, "Unlocking the motorola bootloader," *Azimuth Security Blog*, 2013.

[11] Laginimaineb, "Exploring qualcomm's trustzone," *Bits, Please!*, 2015.

[12] A. Tang *et al.*, "CLKSCREW: Exposing the perils of security-oblivious energy management," in *26th USENIX Security Symposium (USENIX Security 17)*, (Vancouver, BC), pp. 1057–1074, USENIX Association, 2017.

[13] N. Zhang *et al.*, "Truspy: Cache side-channel information leakage from the secure world on arm devices.," 2016.

[14] B. Kevin *et al.*, "How TrustZone could be bypassed: Side-Channel Attacks on a modern System-on-Chip," in *Wistp'17, International Conference on Information Security Theory and Practice*, (Heraklion, Greece), Sept. 2017.

[15] G. Becker *et al.*, "Test vector leakage assessment (TVLA) methodology in practice," in *International Conference on Mathematics and Computing*, 2013.

[16] Global Platform Device Technology, *TEE Internal Core API Specification*, 2013.

[17] D. Genkin *et al.*, "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, ACM, 2016.

[18] B. L. Welch, "The generalization of student's' problem when several different population variances are involved," *Biometrika*, vol. 34, no. 1/2, pp. 28–35, 1947.

[19] E. Brier, C. Clavier, and F. Olivier, "Correlation power analysis with a leakage model," in *International workshop on cryptographic hardware and embedded systems – CHES 2004*, pp. 16–29, Springer, 2004.