REVOLVER: A Zero-Step Execution Emulation Framework for Mitigating Power Side-Channel Attacks on ARM64

Christos Zonios, Vasileios Tenentes
Dept. of Computer Science and Engineering, University of Ioannina, Greece. Emails: {c.zonios, tenentes}@uoi.gr

***Abstract*—Software and hardware vulnerabilities to power side-channel attacks (SCA) are hard to detect and mitigate in systems already deployed in-the-field, because they require specialized equipment and aligned power traces. In this paper, we present REVOLVER, a software-based framework that performs zero-step execution emulation and generates power traces with instruction-level resolution. REVOLVER is a hybrid emulator, because part of it runs on the system that it emulates, an actual ARM64 platform, and evaluates the power consumption of its emulated instructions using actual measurements from on-chip low-frequency power sensors. Such sensors are already present on many system-on-chips (SoCs). To improve the accuracy of the collected traces, REVOLVER repeats the execution of the instructions in a zero-step fashion. To demonstrate the capabilities of our framework, we show that AES keys can be recovered by Correlation Power Analysis (CPA) on traces acquired using REVOLVER, which proves experimentally that there is a leaking power side-channel in the examined system that could potentially be exploited by power SCAs. Moreover, we show how REVOLVER can be used by a security engineer not only to identify software and hardware vulnerabilities to power SCAs, but also to design and evaluate mitigation strategies.**

***Index Terms*—power vulnerabilities, side-channel attack, hardware security, cryptography, mobile devices**

I. Introduction

Microprocessors are deployed in a variety of secure systems that must operate under strict security and privacy standards, such as supply chain monitoring [1] and privacy information management systems [2], to name a few. In order to meet such strict protection requirements, a variety of cryptographic algorithms is widely used [3] to achieve confidentiality and integrity of information during processing and communications. However, software and hardware vulnerabilities are frequently exploited by malicious agents to gain access to privileged execution modes and uncover secrets, undermining system security and user privacy.

In order to protect cryptographic operations against privileged attackers and avoid privilege escalation, CPU designers have integrated hardware for Trusted Execution Environments (TEEs), such as the Intel Software Guard Extensions (SGX) [4] on x86 platforms, and TrustZone [5] on Arm devices. TEEs guarantee the confidentiality and integrity of protected code and data against malicious attacks that may run not only on unprivileged, but also on privileged execution mode, such as a malicious Operating System or a malicious driver.

A widespread type of attack for revealing information is the side-channel attack (SCA). SCAs first appeared during WWII [6] and exploit information inherently carried by the physical implementation of the system that executes the cryptographic algorithms. In the case of a microprocessor, such physical quantities exploited by SCAs have been power consumption [7], electromagnetic radiation [8], time delays [9], faults [10] and sound [11], to name a few. In the past, SCAs uncovered confidential information about user location [12] and keystrokes [13], and were used for privilege escalation by revealing cryptographic keys [14]–[16]. This paper focuses on identifying system vulnerabilities to power SCAs.

Power SCA techniques start by collecting power traces, usually with external equipment, of the cryptographic operation (encryption or decryption). The Simple Power Attack (SPA) is based on the visual inspection of the collected power traces, but it can be subjective, tedious, and is easily mitigated. Automated power SCA methods, such as Differential Power Analysis (DPA) [7] and Correlation Power Analysis (CPA) [17], have been developed that statistically process collected power traces and utilize binary clusters, and Hamming Weight (HW) with Pearson correlation, respectively, to reveal keys.

Despite the protection of cryptographic execution inside a TEE against direct privileged execution, software-based SCAs also exist for TEEs [18]–[20]. Software-based SCAs do not require external measuring equipment for obtaining power traces. On Intel SGX, the PLATYPUS [20] power SCA reveals keys by utilizing low-frequency (1 kHz up to ≈20 kHz) power measurements available by the Intel Running Average Power Limit (RAPL) API and single- and zero-step (single instruction and same instruction execution) execution capabilities of SGX. On TrustZone, the TruSense SCA [19] reveals keys using timing information of cache events, and the Load-Step SCA [18] uses timing information of microarchitectural events obtained at instruction level resolution using a single-step execution mechanism engineered by periodic interrupts from an attacker core to a victim core, as shown in Fig. 1.

To the best of our knowledge, no published software-based power SCA exists, similar to PLATYPUS [20], on TrustZone. We argue that a power SCA on TrustZone might already exist and be deployed in-the-field on actual systems by malicious

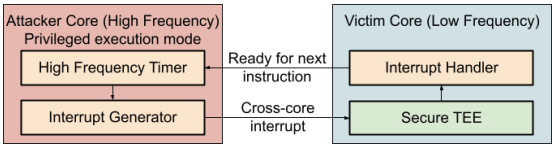

Fig. 1: Threat model against TrustZone TEE [18]

threat agents, even though such SCA is not published. There are, however, already published vulnerabilities, such as the single-step execution [18], showing that a software-based SCA on TrustZone is feasible. As power SCAs are not easily detected, a framework capable of identifying software and hardware security vulnerabilities of a system to power SCAs could be very beneficial to security engineers.

In this paper, we propose REVOLVER, a novel software-based framework that performs zero-step execution emulation on ARM64 and generates power traces with instruction-level resolution. REVOLVER is a hybrid emulator, because part of it runs on the system that it emulates, an actual ARM64 platform, and evaluates the power consumption of its emulated instructions using actual measurements from hardware, by sampling low-frequency on-chip power sensors already present on many Arm-based chips. By performing zero-step execution emulation, the proposed framework collects power traces with instruction-level resolution. Using REVOLVER, we achieve not only to identify software and hardware vulnerabilities of the examined system, an actual ARM System-on-Chip (SoC) running the Advanced Encryption Standard (AES) cipher, but we also develop a mitigation method. The remainder of this paper is structured as follows:

In Sec. II, we collect power traces of various instructions, operands and data, and we prove experimentally that there is a power side-channel on the examined SoC. In Sec. III, by examining software implementing the AES cipher, we demonstrate how REVOLVER can identify software vulnerabilities to power SCAs. In Sec. IV, we show how a security engineer can utilize REVOLVER to reveal hardware vulnerabilities of the examined SoC, and to design and evaluate appropriate mitigation strategies. In Sec. V, we conclude the paper.

II. Single Instruction Power Characterization

A. Platform

We conduct our experiments on the Arm Juno r2 vendor-neutral development board which implements 64-bit Arm architecture AArch64 (Armv8-A) and runs Debian Linux with Kernel version 4.9.0-11. The board operates with an Arm big.LITTLE SoC similar to those found in many mobile devices. The SoC integrates two CPU clusters: the "big" cluster contains two high-performance Cortex-A72 cores, and the "LITTLE" cluster contains four energy-efficient Cortex-A53 cores. The SoC also integrates a GPU. This platform features integrated Low-frequency Power Sensors (LPS) which are accessible by software running on the cores through direct memory access registers. The registers measure current, voltage and power, as well as cumulative energy consumption, for each of the following parts of the SoC: the System (outside of the clusters and GPU), the Cortex-A72 cluster, the Cortex-A53 cluster, and the GPU. The maximum sampling frequency of LPS in our setup is $10\,\mathrm{kHz}$.

B. Single Instruction Power Characterization Process

Since modern processors such as the Cortex-A72 in our setup operate in GHz frequencies, it is obvious that the resolution of the power traces obtainable through LPS is not sufficient to capture single instruction power features. To alleviate this issue, we execute the single Instruction Characterization Process (ICP) of REVOLVER, shown in Fig. 2. First, we repeat N_{reps} times the execution of the instruction that we want to characterize with its corresponding operand values and data (Fig. 2(a)), while continuously measuring power consumption through LPS. For accessing the LPS registers a small */dev/mem* based access driver is implemented that runs in privileged execution mode (Fig. 2(b)). Note that in the examined threat model for TEEs, the attacker runs at privileged execution mode (Fig. 1). When an instruction is characterized on a Cortex-A53, the driver is executed on Cortex-A72, and vice-versa. As the two clusters exist on different power domains, this way we ensure that the LPS driver execution is not intrusive to our measurements. After the execution of the instruction loop, the average power consumption of the instruction under characterization is obtained by sampling the cumulative energy sensors. It is $P_{ins} = E_{cum}/t_N$, where P_{ins} is the average instruction power, E_{cum} the cumulative energy measured by LPS and t_N the time needed for the loop execution, which is measured using CPU clock cycle registers. Similarly, we obtain the energy per instruction E_{ins} using: $E_{ins} = E_{cum}/N_{reps}$. This process circumvents the problem of the low sampling frequency of LPS and negates noise of other running system processes (details in Sec. IV).

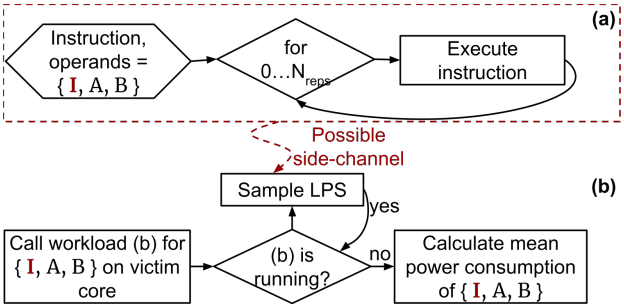

Fig. 2: REVOLVER ICP: (a) instruction execution runs on victim core; (b) LPS sampled on attacker core

C. Distinguishing Instructions and Data

First, we demonstrate that LPS energy measurements collected through the REVOLVER ICP are sufficient to distinguish an instruction. Although the ICP can be applied to any instruction type, we focus on a set of four instructions {UXTB, MOV, STRB, LDRB} that exist in the AES cryptographic algorithm implementation, which is discussed in Sec. III. We collect data on a Cortex-A72 core, using three different registers and all possible operand data values. Fig. 3(a) depicts the energy per instruction for instructions UXTB and MOV, which are applied on register operands. Fig. 3(b) depicts the energy per instruction for instructions STRB and LDRB, which are memory store and load byte instructions. Each instruction is repeated $N_{reps} = 1$ billion times and LDRB and STRB are given fixed address values and the same registers.

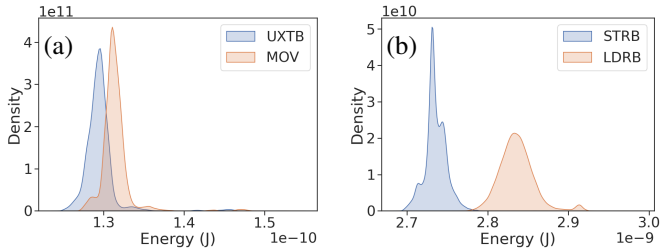

Fig. 3: Energy per instruction distributions for different instruction types: (a) byte instructions, (b) memory instructions

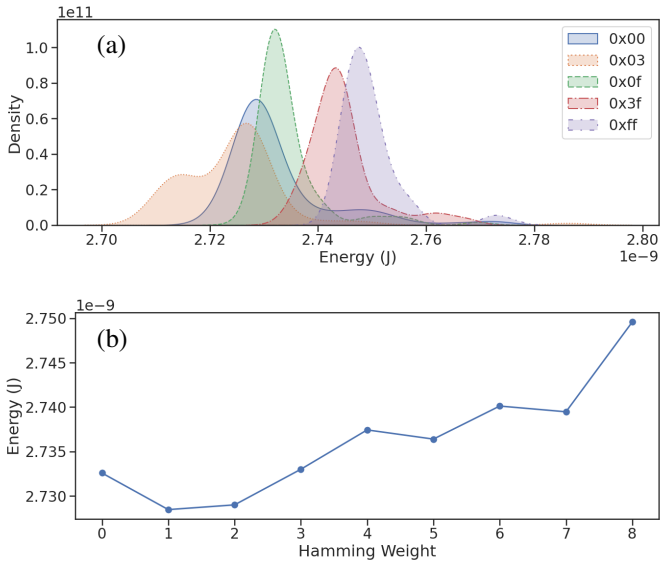

Fig. 4: Energy per STRB instruction: (a) for different operands, (b) for different Hamming Weights of the operand

We observe that the energy per instruction distributions of all examined instructions are clearly distinguishable.

In Fig. 4, we examine how the energy consumption of a single instruction is affected by operand values. We measure the energy distribution per instruction for five instances of instruction STRB (Fig. 4(a)). We maintain the memory address where data is stored constant, and we examine five different values of 8 bits (1 byte) for each instance. It is evident that operand values can be distinguished by energy measurements obtained by REVOLVER ICP. In Fig. 4(b), we present the average energy per STRB instruction instance, while sweeping the value stored in memory. Values are sorted by their Hamming Weight, i.e. the number of logic-'1' bits. The near-monotonic line obtained is an indication that execution information is leaking through core power.

The same observations have been verified on every core of both clusters in the SoC. These results show that both instructions and operands can be distinguished by our framework. This is also a strong indication that information is leaking through the power sensors, and one of the hardware security vulnerabilities in our system identified by REVOLVER.

III. Proposed Zero-Step Execution Emulation

In this section, we present the proposed zero-step execution emulation process of REVOLVER. We will demonstrate how this process allows for the identification of system security vulnerabilities and the evaluation of mitigation techniques, using an AES implementation, as a proof-of-concept example.

A. Preliminaries on Advanced Encryption Standard (AES)

The Advanced Encryption Standard (AES) is a symmetric cipher, which uses the same private key to encrypt and decrypt information, and is shown in Fig. 5. AES encrypts 128 bits (16 bytes) of plaintext P into 128 bits of ciphertext C using a private key. The decryption process is the inverse of the encryption, as all operations are reversible with the same key. AES comes in three variants, according to its key length: AES-128, AES-192, and AES-256.

The encryption process contains several rounds of the same operations involving the key, the plaintex P and AES internal state S. K, P and S are organized in 4×4 tables (shown in Fig. 6). Next, the AES rounds are described:

SubBytes: a look-up-table is used to substitute bytes on S.

ShiftRows: on each row, columns are shifted; first row: no shift, second row: shift left by one, third row: shift left by two, fourth row: shift left by three (wraparound).

MixColumns: each column of the grid is replaced by its dot product with a fixed matrix, defined in the standard. The add operation of the dot product is performed using XOR, and a finite field multiplication is performed.

AddRoundKey: each key byte is added to the proper byte of S using XOR after MixColumns is applied. AddRoundKey is represented with the $\oplus$ symbol in Fig. 5.

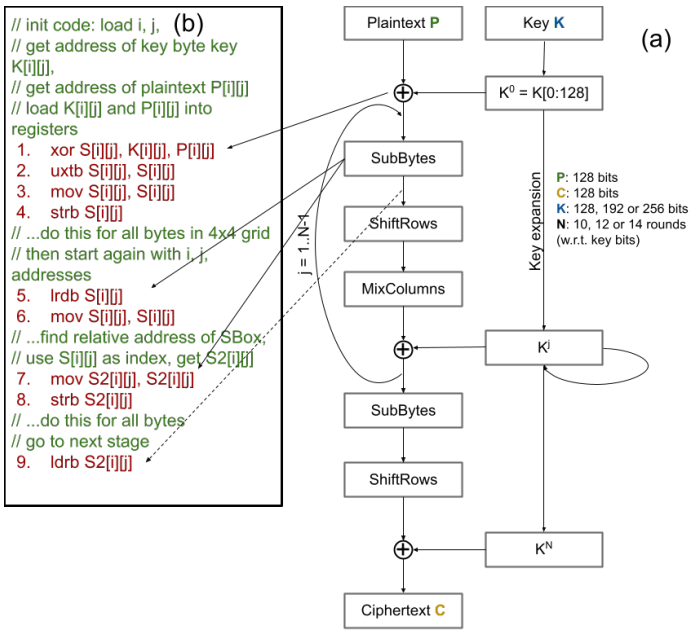

Fig. 5: (a) Overview of the AES encryption process; (b) AES pseudo-assembly, highlighting instructions involving key

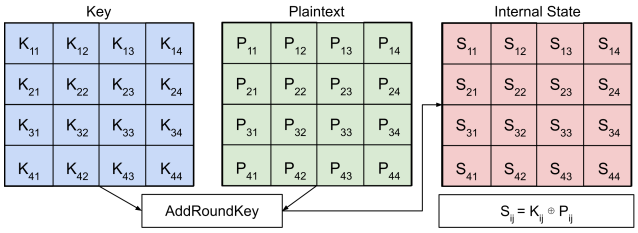

Fig. 6: During the AddRoundKey operation, each byte S_{ij} is calculated using only bytes K_{ij} and P_{ij}

B. Zero-step execution emulation process

In Fig. 7, we present a process, hereafter denoted as *zero-step execution emulation*, which is applied on a sequence of assembly instructions, which consist the *emulated program* (EP), shown in Fig. 7(a). The process is separated into two sub-processes. First, is the *zero-step ICP sub-process* (Fig. 7(b)), during which REVOLVER ICP is used to characterize instructions I and operand values A, B. Pre-characterized $\{I, A, B\}$ combinations are stored in a database. The pre-characterization of the instructions is not mandatory as REVOLVER ICP can be called on-the-fly on known run-time values. Next is the *execution emulation* sub-process (Fig. 7(c)), during which the sequence of input instructions is emulated in order to compute their run-time operand values. Note that this sub-process may run on a different system than the one that is emulated. The power trace T_{EP} of the emulated program is generated by retrieving and concatenating the power of all the instructions (or a subset), from the input sequence, retrieved from the database using its known run-time operand values.

C. Emulating an AES implementation using REVOLVER

SCAs on AES often exploit two algorithm vulnerabilities: the first 16 bytes of the private key are used as-is in the first round during encryption (or the last round during decryption), and the addition of the key to the plaintext is on a per-byte basis, as shown in Fig. 6. Therefore a SCA, when applied successfully, targets key bytes one-by-one (16×2^8 guesses) instead of targeting the whole key (2^{128} guesses) at once. In Fig. 5(b), the instructions of the first round AddRoundKey and SubBytes operations that use the key and plaintext data are depicted. If a subkey guess is correct, the power traces and the calculated output of operation SubBytes are correlated.

We apply REVOLVER (Fig. 7) to emulate the run-time values of operands, but we generate traces only for the instructions shown in Fig. 5(b). The set of instructions is small, therefore a database of instruction characterizations for all possible run-time operand values can be generated with the zero-step ICP sub-process (Fig. 7(b)). After characterization, it is feasible to generate a power trace for any key/plaintext combination of run-time values. An example of constructed traces for a specific plaintext and two different keys is shown in Fig. 8. Despite the differences of the traces, there are not enough to identify keys by visually inspecting them. In the next section, we examine a statistical SCA method.

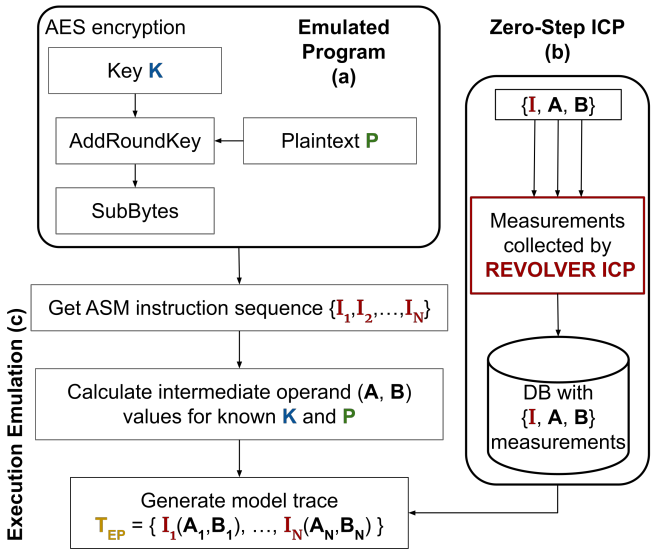

Fig. 7: Zero-step execution emulation applied on AES.

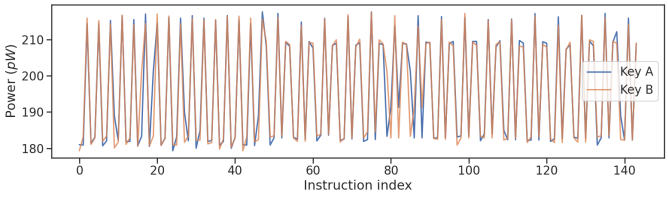

Fig. 8: Power traces from AES zero-step execution emulation.

IV. Vulnerabilities Identification and Mitigation

A. Software security vulnerabilities: AES Key Recovery

A statistical SCA method, such as CPA [17], requires a large set of power traces T to reveal a secret key K. A trace is obtained during the encryption (or decryption) of a known plaintext P using the secret key K. In our experiments, we generate each trace in T using zero-step execution emulation (Fig. 7) of the AES-128 encryption of a plaintext with a secret key K. The number of plaintexts used for power traces collection, which is equal to the size of set T, is denoted as N_P. Next, we apply CPA on T in order to reveal the secret key K used during the collection of power traces.

In Table I, we summarize the variables that affect the generated power traces and the ability of CPA to reveal keys. N_{reps} is the number of times an instruction is repeated during REVOLVER ICP (Fig. 2). To tune N_{reps}, we perform an experiment where we use various values in the range $[10M, 1B]$ and examine the percentage of key bytes guessed correctly, as shown in Fig. 9. The plaintexts number N_P used to collect power traces is examined in the range $[0.1K, 10K]$. Parameter M denotes the power sensor observed during zero-step execution emulation. The results in this section are obtained considering a Cortex-A72 victim core and collecting measurements from the Cortex-A72 cluster power sensor.

TABLE I: Parameters affecting CPA efficiency

Param	Description	Value
N_P	Number of 16-byte plaintexts to use. Granularity of traces set T	0.1K up to 10K
N_{reps}	Number of times each instruction is repeated during REVOLVER ICP	10M up to 1B
M	Utilized LPS	$P_{sys}, P_{A72}, P_{A53}, P_{GPU}$

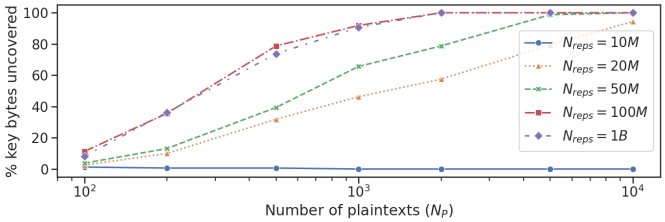

Fig. 9: CPA efficiency achieved for various N_{reps} and number of plaintexts N_P. Obtained through Cortex-A72 power sensor.

Fig. 9 shows how the efficiency of CPA on uncovering a secret key K is affected by these parameters. The $x-$axis depicts the number of plaintexts (and power traces) N_P examined. The y-axis depicts *CPA efficiency*, which is evaluated as the the percentage of key bytes revealed successfully out of the 16 bytes of the AES-128 key. Each line is the CPA efficiency for a different value of the N_{reps} parameter. The experiment is repeated for 10 different randomly generated keys K, and the percentage is calculated over all keys. We observe that the number of times an instruction is repeated during characterization (N_{reps}) affects the performance of CPA, which is attributed to the power per instruction accuracy improvement achieved by ICP (Fig. 2). We also observe, as expected, that as the number of plaintexts N_P increases so does the performance of CPA. In fact using only $N_P \geq 2000$ plaintexts and $N_{reps} = 1B$ the keys are 100% successfully revealed. Similarly, the keys are 100% revealed using $N_P \geq 5000$ plaintexts and $N_{reps} = 50M$.

B. Identifying hardware security vulnerabilities

So far we have presented results considering a Cortex-A72 victim core. Next, we examine different cores, but also different power sensors. The CPA efficiency is shown in Table II for two Cortex-A72 (columns 2, 3, 4 and 5) and two Cortex-A53 (columns 6, 7, 8 and 9) cores, using all power sensors. The first column contains the LPS used (parameter M) from the available power sensors (details presented in Sec. II-A). Each row shows the CPA efficiency achieved when the collected traces are obtained from the corresponding power sensor. Parameter values are: $N_{reps} = 1B$ and $N_P \in \{500, 2000\}$. Interestingly, we observe that by utilizing the GPU power sensors, all subkeys are revealed successfully, which means that there is a power side-channel leaking information from the power domain of the cores towards the power domain of the GPU, which is an actual hardware vulnerability of the examined system identified by REVOLVER.

TABLE II: CPA efficiency for different cores and sensors

LPS	A72 (1)		A72 (2)		A53 (1)		A53 (2)	
$N_P \longrightarrow$	500	2000	500	2000	500	2000	500	2000
P_{SYS}	16/16	16/16	10/16	16/16	15/16	11/16	0/16	0/16
P_{A72}	16/16	15/16	6/16	16/16	0/16	4/16	0/16	0/16
P_{A53}	16/16	16/16	0/16	0/16	9/16	0/16	1/16	2/16
P_{GPU}	16/16	16/16	16/16	16/16	14/16	16/16	16/16	16/16

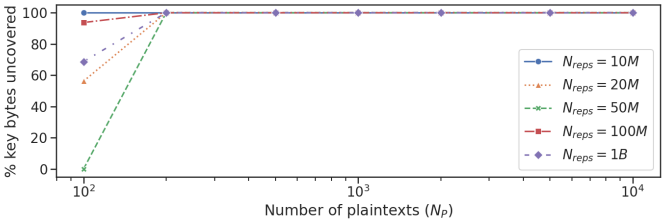

Fig. 10: CPA efficiency achieved for various N_{reps} and number of plaintexts N_P. Obtained through GPU power sensor.

After identifying this hardware vulnerability, we repeat the experiment presented in Fig. 9, but this time we use the GPU power sensor instead of that of the Cortex-A72 cluster, although the victim core is a Cortex-A72 core. The results are depicted in Fig 10. Interestingly, we observe that CPA efficiency reaches 100% using less than 200 plaintexts for all examined N_{reps} values. Note that only values from the GPU LPS sensor are used to construct the set of power traces T processed by CPA. Clearly, there is a side-channel between AES executed on the Cortex-A72 cluster and the GPU. We attribute this side-channel to the cache coherency circuitry available between the CPU clusters and the GPU in our system. Using the REVOLVER framework, a security engineer can identify such hardware vulnerabilities to power SCAs and decide to deactivate (through the device tree) such power sensors from a system or allow access to these sensors only by trusted firmware.

C. Mitigation of vulnerabilities

In this paragraph, through a proof-of-concept example, we show how a security designer can utilize the proposed framework to design and evaluate their power SCA mitigation strategies. REVOLVER instruction-resolution power traces allow for evaluating the instructions in our AES implementation that contribute more towards revealing secrets by comparing the maximum Pearson correlation coefficients of all instructions in the power traces. In Fig. 11, we present these coefficients for the correct key byte and four random incorrect key bytes. Some instructions in the AES implementation are more revealing than others. A security engineer can identify these vulnerable instructions using the proposed REVOLVER framework in order to design and evaluate an appropriate mitigation strategy.

Our mitigation against the power SCA is based on arbitrary computations re-ordering for the AES implementation. We consider the AES AddRoundKey operation shown in Fig. 6. In the implementation of AES used in our experiments, each byte of the internal state S is calculated in order, from

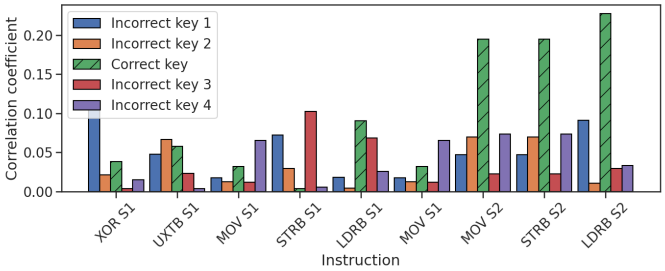

Fig. 11: Pearson coefs. Correct vs incorrect AES key bytes.

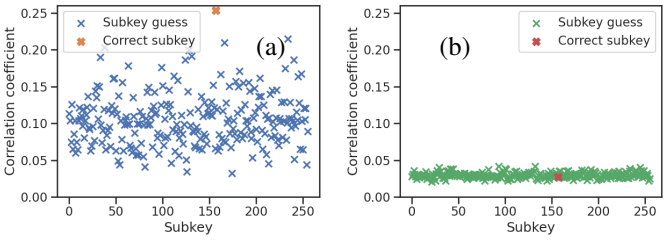

Fig. 12: Pearson coefs. for all guesses of first AES key byte: (a) without; and (b) with mitigation. N_{reps}=1B N_P=10K.

$S_{11}, S_{12}, \ldots, S_{44}$. The mitigation strategy is based on computing the internal state bytes in an arbitrary order, therefore leading to arbitrary mixing the power traces of the last 3 instructions, which reveal the key (Fig. 11). In Fig. 12, we present the results before (Fig. 12 (a)) and after (Fig. 12 (b)) introducing our mitigation method, for a single AES key byte. Each point is the maximum correlation coefficient of the subkey guess byte with the set of traces T (the higher the correlation, the more likely the guess is correct). The correct subkey is annotated in red. This technique eradicates the correlation between traces and subkey, thus mitigating the SCA on the obtained traces.

V. Conclusion

We presented REVOLVER (Figs. 2 and 7), a software-based framework that performs zero-step execution emulation on ARM64 and generates power traces with instruction-level resolution (Fig. 8). REVOLVER is a hybrid emulator, because part of it runs on the system that it emulates and evaluates the power consumption of its emulated instructions using measurements from on-chip power sensors that are already present on many Arm-based chips. To improve the accuracy of the collected traces, REVOLVER repeats the execution of the instructions in a zero-step fashion. We demonstrated that AES keys can be recovered by REVOLVER traces (Figs. 9, 10 and Table II) using Correlation Power Analysis (CPA), which proves experimentally that there is a leaking power side-channel in the examined system that could potentially be exploited by power SCAs. Furthermore, we demonstrated how REVOLVER can be used by a security engineer to identify system vulnerabilities to power SCAs (Figs. 3, 4 and 11), and to design and evaluate mitigation strategies (Fig. 12).

Acknowledgements

This work is co-financed by the European Regional Development Fund of the European Union and Greek national funds through the Operational Program "Competitiveness, Entrepreneurship and Innovation", under the call RESEARCH–CREATE-INNOVATE (project code: T2EDK-02836). The authors would like to thank Arm Research for donating them the platform for conducting research on mitigation mechanisms for power-supply side-channel attacks.

References

[1] "Specification for security management systems for the supply chain," Standard ISO 28000:2007, 2007.

[2] "Security techniques — extension to iso/iec 27001 and iso/iec 27002 for privacy information management — requirements and guidelines," Standard ISO/IEC 27701:2019, 2019.

[3] R. L. Rivest, *CHAPTER 13 - Cryptography*, ser. Handbook of Theoretical Computer Science. Elsevier, Jan 1990, p. 717–755.

[4] Intel, "Intel® software guard extensions (intel® sgx)."

[5] A. Holdings, "Arm security technology: Building a secure system using trustzone technology," *Retrieved on June*, vol. 10, p. 2021, 2009.

[6] N. TEMPEST, "A Signal Problem," *Cryptologic Spectrum*, vol. 2, 1972.

[7] P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," in *Advances in Cryptology — CRYPTO' 99*, M. Wiener, Ed. Berlin, Heidelberg: Springer, 1999, pp. 388–397.

[8] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi, "The EM Side—Channel(s)," in *Cryptographic Hardware and Embedded Systems - CHES 2002*. Berlin, Heidelberg: Springer, 2003, pp. 29–45.

[9] A. Bortz and D. Boneh, "Exposing private information by timing web applications," in *Proceedings of the 16th international conference on World Wide Web*, ser. WWW '07. New York, NY, USA: Association for Computing Machinery, May 2007, pp. 621–628.

[10] E. Biham and A. Shamir, "Differential fault analysis of secret key cryptosystems," in *Advances in Cryptology — CRYPTO '97*, B. S. Kaliski, Ed. Berlin, Heidelberg: Springer, 1997, pp. 513–525.

[11] A. Das, N. Borisov, and M. Caesar, "Do You Hear What I Hear? Fingerprinting Smart Devices Through Embedded Acoustic Components," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '14. New York, NY, USA: Association for Computing Machinery, Nov. 2014, pp. 441–452.

[12] Y. Michalevsky, A. Schulman, G. A. Veerapandian, D. Boneh, and G. Nakibly, "PowerSpy: Location tracking using mobile device power analysis," in *24th USENIX Security Symposium (USENIX Security 15)*. Washington, D.C.: USENIX Association, Aug. 2015, pp. 785–800.

[13] K. S. Killourhy and R. A. Maxion, "Comparing anomaly-detection algorithms for keystroke dynamics," in *2009 IEEE/IFIP International Conference on Dependable Systems Networks*, Jun. 2009, pp. 125–134.

[14] K. Ramezanpour, P. Ampadu, and W. Diehl, "SCAUL: Power Side-Channel Analysis With Unsupervised Learning," *IEEE Transactions on Computers*, vol. 69, no. 11, pp. 1626–1638, Nov. 2020.

[15] P. Socha, V. Miškovský, H. Kubátová, and M. Novotný, "Correlation Power Analysis Distinguisher Based on the Correlation Trace Derivative," in *2018 21st Euromicro Conference on Digital System Design (DSD)*, Aug. 2018, pp. 565–568.

[16] L. Batina, B. Gierlichs, and K. Lemke-Rust, "Differential Cluster Analysis," in *Cryptographic Hardware and Embedded Systems - CHES 2009*. Springer, 2009, pp. 112–127.

[17] E. Brier, C. Clavier, and F. Olivier, "Correlation Power Analysis with a Leakage Model," in *Cryptographic Hardware and Embedded Systems - CHES 2004*. Springer, 2004, pp. 16–29.

[18] Z. Kou, W. He, S. Sinha, and W. Zhang, "Load-step: A precise trustzone execution control framework for exploring new side-channel attacks like flush+evict," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, Dec 2021, p. 979–984.

[19] N. Zhang, K. Sun, D. Shands, W. Lou, and Y. T. Hou, "TruSense: Information Leakage from TrustZone," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, Apr. 2018, pp. 1097–1105.

[20] M. Lipp, A. Kogler, D. Oswald, M. Schwarz, C. Easdon, C. Canella, and D. Gruss, "PLATYPUS: Software-based Power Side-Channel Attacks on x86," in *2021 IEEE Symposium on Security and Privacy (SP)*, May 2021, pp. 355–371, iSSN: 2375-1207.