Exploiting Bus Communication to Improve Cache Attacks on Systems-on-Chips

Johanna Sepulveda*, Mathieu Gross*, Andreas Zankl†, and Georg Sigl*†
*Technische Universität München, Germany
johanna.sepulveda@tum.de, mathieu.gross@tum.de, sigl@tum.de
†Fraunhofer Institute AISEC, Germany, andreas.zankl@aisec.fraunhofer.de, georg.sigl@aisec.fraunhofer.de

***Abstract*—Systems-on-Chips (SoCs) are one of the key enabling technologies for the Internet-of-Things (IoT). Given the continuous distribution of IoT devices, data confidentiality and user privacy are of utmost importance. However, with the growing complexity of SoCs, the risk of malware infections and trojans introduced at design time increases significantly. A vital threat to system security are so-called side-channel attacks based on cache observations. While mainly studied on desktop and server systems, recent publications have analyzed cache attacks on mobile devices and network-on-chip platforms. In this work, we investigate cache attacks on System-on-Chips implementing bus based communication. To this end, we present two contributions. First, we demonstrate an improved Prime+Probe based cache attack on AES-128 that, for the first time, exploits the bus communication to increase its efficiency. Second, we integrate two countermeasures (Shuffling and Mini-table) and evaluate their impact on the attack. The results show that our improved attack recovers the full key twice as fast as Prime+Probe without exploiting bus communication. Moreover, we propose protection techniques that are feasible and effectively mitigate both original and improved attack.**

***Index Terms*—SoCs, Cache Attack, Access-driven, Bus, Security.**

I. Introduction

The Internet-of-Things (IoT) is becoming one of the keystones for the information and communication technology market. IoT endpoints are usually able to sense the environment, pre-process data and communicate information. Most of these endpoints are implemented as Systems-on-Chips (SoCs), which are considered one of the key enabling technologies for the IoT. SoCs are comprised of processing units, memories and peripherals (provided as Intellectual Property, or IP, cores) and communication components used for data exchange among the IP cores.

High VLSI integration levels and demanding requirements of the current embedded applications promote the development and widespread adoption of complex SoCs, such as Multi-Processor System-on-Chips (MPSoCs). These devices, also known as *system of systems*, are organized in a tile-based architecture interconnected through a Network-on-Chip (NoC) using routers and links to exchange data. Each tile is composed of a single IP hardware core (e.g. single processor or memory) or of a cluster of IP hardware cores (several processors and shared memories), which communicate through a bus. In order to increase the efficiency and flexibility of those systems, memory hierarchies up to L3 caches and DRAMs are integrated as well. Communication among tiles is performed through the NoC. Fig. 1 shows an example of a 9-tile MPSoC interconnected through a 9-router (R_1 to R_9) mesh-based NoC. Tile 6 is composed of three IP cores (IP_1, IP_2 and IP_3), where IP_3 is a shared memory. Those components communicate through a bus. The router that links tile 6 to the other components of the MPSoC is R_6. This architecture is compliant with current state of the art market MPSoCs such as Tile-Mx100 from Tilera [1], MPPA from Kalray [2] or SCC from Intel [3].

MPSoCs operating in the context of IoT usually integrate security features such as cryptographic hardware cores for confidentiality and authentication security services. However, these components are prone to implementation attacks. During the operation of a cryptographic core, the secret key may passively be revealed through so-called side-channels. Classical side-channels include the measurement of the execution time, power-consumption and electromagnetic (EM) radiation of the cryptographic IP core. The interconnection of MPSoCs operating in the Internet-of-Things has shifted the attention of attackers to another kind of side-channel attacks: microarchitectural side-channels that emerge from sharing resources on the MPSoC.

Cache hierarchies are a common target in microarchitectural attacks. In general, caches are shared resources designed to speedup the execution of applications, including the execution of cryptographic operations. However, they also allow processes to mutually interfere with each other, which becomes a major threat on MPSoCs, if malicious and sensitive processes are executed together. Attacks are commonly categorized based on the side-channel information available to the adversary: execution time (time-driven attacks [4] [5]), sequences of cache operations like hits and misses (trace-driven attacks [6]), and cache access patterns (access-driven attacks [7]).

Cache attacks have been thoroughly studied on desktop and server systems. Only recently it was found that cache

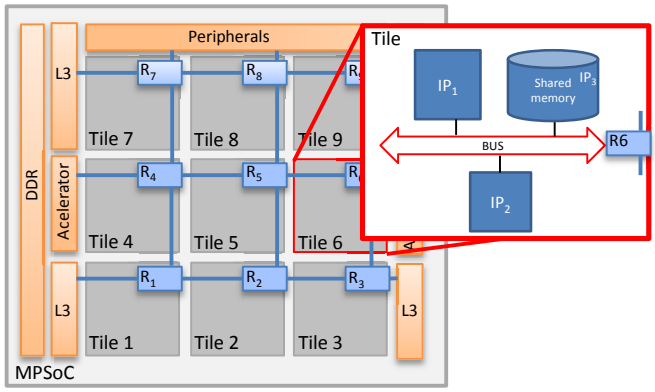

Fig. 1: Tile-based MPSoC architecture.

2159-3477/17 $31.00 © 2017 IEEE
DOI 10.1109/ISVLSI.2017.57

attacks can also be executed in SoC environments [8] [9] [10] [11] [12]. Exploiting communication to optimize cache attacks is proposed in [11] [12]. The authors monitor the NoC to decrease the amount of traces required to successfully conduct the attacks. As a countermeasure, the authors propose a notification-based protocol that is integrated into the network routers. Despite the promising results, previous work exhibits three drawbacks: i) NoC-based attacks are not effective in tile-based MPSoC organizations, because cryptographic cores and caches are contained in a single tile, avoiding any network data exchange; ii) the monitor mechanism requires an elaborated calibration due to the complexity of the NoC; iii) protection mechanisms are not scalable due to the communication overhead required for the continuous surveillance.

To overcome these drawbacks, we propose the first access-driven cache attack targeting a table based implementation of the Advanced Encryption Standard (AES) that is integrated into a bus-based SoC. The attack can be performed in simple bus-based SoCs or in tile-based MPSoCs as shown in Fig. 1. We also implement two countermeasures: Shuffling and Mini-tables. In summary, the contributions of the paper are:

- the implementation of an access-driven attack on AES in a bus-based MPSoC,
- the exploitation of the bus communication to increase the attack efficiency,
- the integration of two protection mechanisms: Shuffling and Mini-tables, and
- the practical evaluation of the improved attack and the protection mechanisms including a comparison with previous work.

The rest of the paper is organized as follows. Section II presents the previous work on access-driven cache attacks. Section III describes the bus-based MPSoC and the threat model. Section IV describes the improved attack and Section V presents its execution on the bus-based MPSoC. The countermeasures and their evaluation are presented in Section VI, before the paper concludes in Section VII.

II. Related Work

The first extensive analysis of access-driven cache attacks on transformation table (T-table) based AES implementations is provided by Osvik et al. [7]. The authors propose two attack techniques: Evict+Time and Prime+Probe. Both techniques require an adversary to remove, i.e., evict cache lines storing T-table entries from the cache hierarchy and to test whether the AES process used the evicted lines during encryption. With the relations between cipher input and secret key, the T-table access information allows the adversary to recover the entire key. By using Prime+Probe on an AMD Athlon 64, the authors retrieve a full AES-128 key with a second-round attack after 300 encryptions. Neve and Seifert [13] investigate key recovery based on non-accessed T-table entries in the last round of AES. The authors conclude that less than 20 encryptions can be sufficient to recover the entire key in a last round attack. However, this requires that the AES implementation uses a special table in the last round of AES, which is not always fulfilled. Zhao et al. [14] propose a second-round Prime+Probe attack on AES that is able to retrieve a full AES-128 key from the non-accessed T-table entries of 80 encryptions. Irazoqui et al. [15] propose extensions to the original Prime+Probe attack that enable it in virtualized environments. They allow an adversary to recover a full AES key across virtual machine boundaries with $2^{18.93}$ encryptions using a last round attack.

While most access-driven attacks have been implemented on desktop and server systems, some publications have studied attacking AES on embedded System-on-Chips. Spreitzer and Plos [16] implement an access-driven attack based on Evict+Time on an ARM Cortex-A8 SoC. The authors demonstrate that table misalignment can significantly improve the attack success rate. Still, at least $2^{16.91}$ encryptions are necessary to recover the entire key. Lipp et al. [8] demonstrate that several access-driven attack techniques, including Prime+Probe, are generally applicable to ARMv7 and ARMv8 compliant processors. However, the authors do not provide the exact number of encryptions necessary for full key recovery using Prime+Probe.

Communication leakage in SoCs has been pointed out for the first time by Sepúlveda et al. [17]. Based on this concept, Reinbrecht et al. [18] implement a Prime+Probe based cache attack on a NoC-MPSoC with an ARM Cortex-A9. The authors exploit contentions in the network communication to detect the end of the first AES round. The Probe step is then performed immediately after the first round to improve the quality of the cache observations. With this approach, the authors are able to recover 12 of the 16-byte AES key after 80 encryptions. In response to the proposed attack, Reinbrecht et al. [11] and Fernandes et al. [12] propose security-aware traffic management and routing protocols, respectively.

In contrast to previous work, we implement a Prime+Probe based cache attack in a bus-based MPSoC, where approaches related to NoCs do not apply. We show how to exploit the bus communication to also find the end of the first AES round and immediately start the Probe step. In contrast to Reinbrecht et al. [18], we achieve full key recovery after only 40 encryptions.

III. Description of the Bus-based MPSoC

MPSoCs are tile-based structures which are able to meet a variety of application demands. As shown in Fig. 1, each tile is either composed of a single IP core or a cluster of IP cores that communicate through a bus. The tile can include heterogeneous processing units, storage components, peripherals, hardware accelerators and other IP hardware cores. Data is exchanged over a NoC between tiles. In order to increase the performance, current SoCs implement memory hierarchies, where several levels of cache (e.g. L1 to L3) and a set of DRAMs are integrated. When a cache miss occurs, all cache levels of a processing core are queried until the requested data is found. If no level contains the data, the cache coherency mechanism initiates an access to other distant cores or eventually to DRAM. The remainder of this section describes the T-table AES implementation and the thread model considered in this work.

A. T-table AES Implementation

The introduction of SoCs in IoT environments promotes the integration of security primitives such as the Advanced Encryption Standard (AES). AES is a symmetric key encryption algorithm designed to meet the demands of the U.S National Institute of Standard and Technology (NIST) [19]. AES encrypts 128 bits of data with key lengths of 128/192/256 bits using 10/12/14 rounds, respectively. In this work, AES-128 is used. AES intermediate states are represented as a

4x4 state matrix. This matrix is operated on iteratively for 10 rounds, each composed of four round operations: AddRoundKey (XORing the state with the current round key), SubByte (byte substitution), ShiftRow (byte transposition) and MixColumn (matrix multiplication).

Performance optimized implementations of AES use transformation tables (T-tables), where the SubByte, ShiftRow and MixColumn operations are reduced to four lookup tables (T0, T1, T2 and T3) whose entries are simply XOR'ed. In this implementation, one round consists of 16 table lookups. Rounds 1 to 9 can be calculated as in Equation (1). The last round of AES does not contain the MixColumn operation and is sometimes implemented with a fifth table T4. Our attack is focused on the first round of AES and therefore does not rely on the existence of T4. The fundamental mechanism exploited by the cache attack is that T-tables are accessed depending on the secret key.

$$\begin{array}{l}(x_0^{r+1}, x_1^{r+1}, x_2^{r+1}, x_3^{r+1}) \leftarrow T_0[x_0^r] \oplus T_1[x_5^r] \oplus T_2[x_{10}^r] \oplus T_3[x_{15}^r] \oplus k_0^{r+1} \\ (x_4^{r+1}, x_5^{r+1}, x_6^{r+1}, x_7^{r+1}) \leftarrow T_0[x_4^r] \oplus T_1[x_9^r] \oplus T_2[x_{14}^r] \oplus T_3[x_3^r] \oplus k_1^{r+1} \\ (x_8^{r+1}, x_9^{r+1}, x_{10}^{r+1}, x_{11}^{r+1}) \leftarrow T_0[x_8^r] \oplus T_1[x_{13}^r] \oplus T_2[x_2^r] \oplus T_3[x_7^r] \oplus k_2^{r+1} \\ (x_{12}^{r+1}, x_{13}^{r+1}, x_{14}^{r+1}, x_{15}^{r+1}) \leftarrow T_0[x_{12}^r] \oplus T_1[x_1^r] \oplus T_2[x_6^r] \oplus T_3[x_{11}^r] \oplus k_3^{r+1}\end{array} \quad (1)$$

In the AES T-Table implementation, the indexes used during the first round are defined as in Equation (2). Each byte i of the intermediate value x involves a plaintext byte P_i, a table lookup T_i (which returns a 4-byte value) and the key K_i. The simple relation of the plaintext to the secret key in this round is exploited in the attack.

$$x_i^0 = P_i \oplus K_i \quad (2)$$

B. Threat Model Description

The system considered in this work is the bus-based subsystem shown in Fig. 1. It consists of a processor IP (IP_1) with a private L1 cache, a shared L2 cache (IP_3) and another processor IP, which is able to read and write L2 IP_2. The victim process implements and executes AES in software using T-Tables on IP_1. During execution, the T-Tables are stored in the memory hierarchy of the system (L1, L2, DRAM). The spy process is a piece of malware executed on IP_2. Spy and victim are therefore physically isolated on different IP cores. The attack assumes that victim and spy processes are executed in parallel. The goal of the spy is to detect whether T-table entries stored on cache lines in L2 have been evicted or not. The spy only uses legitimate memory read and write operations that affect the L2 cache. Moreover, the spy does not have elevated, i.e., administrator privileges. The spy is able to infer accesses to L2 by the victim, because both processes use the same bus to communicate with the L2 cache. Any reduction in the spy's communication throughput reveals an access to L2 by the victim (for details see Section IV). In order to implement this bus-enhanced access-driven cache attack, the following requirements must be fulfilled:

- The attacker can infect an IP core within the SoC.
- The attacker can generate plaintexts and trigger encryptions on IP_1.
- The private L1 cache of the victim contains no T-Tables elements before the encryption.
- The attacker knows or can learn the cache sets corresponding to the T-Tables.

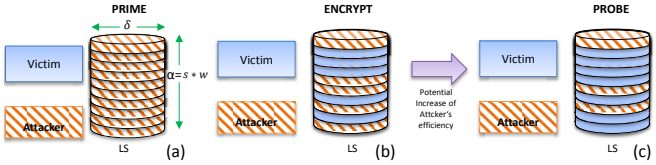

Fig. 2: Prime+Probe based access-driven cache attack.

The infection of a processor IP core is realistic, given the complexity of modern embedded application processors that are able to run full scale operating systems and execute multiple tasks. The generation of plaintexts is a simple operation for hardware or software based (pseudo-random) number generators. Triggering AES encryptions at the victim does not limit the general applicability of our proposed attack. Asynchronous attack scenarios are possible, but require increased acquisition and post-processing effort. However, this does not render the attack infeasible and is therefore omitted. An additional discussion of the requirements and challenges is given in Section V.

IV. Access-driven Cache Attack

The Prime+Probe attack is shown in the Fig. 2. The victim and the spy share a S-associative L2 cache. It has size α, w ways per set, and δ words per cache line. During encryption, the victim's data (T-Tables T0, T1, T2, T3) is stored in L2. Different colors represent the owner of the cache lines (victim or spy). The attack is composed of three steps. In the first step (Prime), shown in Fig. 2a, the spy starts with evicting (or priming) cache sets X. In the second step, the spy requests the victim to encrypt a plaintext, as shown in the Fig. 2b. The encryption process leads to the eviction of some cache lines owned by the spy. After the encryption, in the third step (Probe), the spy re-accesses (or probes) the data previously brought into the cache, as shown in Fig. 2c. The purpose is to detect the accessed cache sets during encryption. If the victim accesses data in X, (parts of) the spys data is removed from cache, producing a corresponding number of cache misses. Thus, it takes the spy longer to retrieve its data. If the victim does not use X, the spy's access to its data is faster as all accesses result in cache hits.

The efficiency of the access-driven cache attack depends on the amount of noise in the spy's measurements. In case of a first round attack, the cache sets not accessed contain useful information for the spy [13]. The cache sets accessed in the subsequent rounds are additional sources of noise which are going to be interpreted as false-positives in the first round attack. For the spy, the largest amount of information is observable after the first 16 T-table accesses to the shared L2 cache. To fully leverage this information, the spy must start to probe L2 after the first round of the AES encryption.

V. Bus-based Attack Improvement

The improvement of the access-driven cache attack is realized by exploiting the bus-based communication. To the best of our knowledge, this has not been proposed and implemented in previous literature. By monitoring the shared bus, it is possible for the spy to detect, in a non-intrusive way, the optimal point in time to probe the shared L2 cache, i.e., the end of the first

round of encryption. This section presents the optimization of the attack, the process for the retrieval of the key and the challenges of this approach.

A. Description

Fig. 3 illustrates the Bus-based system. The figure again shows the victim IP core with its private L1 cache, a shared L2 cache, and a spy IP core (IP_2). These cores exchange data through a shared bus.

This configuration enables the spy to detect the first round of AES by observing the bus. After priming the L2 cache, the spy triggers an AES encryption (1). As a result, the AES implementation accesses the T-tables, thus, issuing data requests to the private L1 cache (2a). The availability of the data in L1 is checked. If the data is not in L1, an access to L2 is performed (2b). Meanwhile, the spy is continuously requesting read accesses to L2 (3), which allows the spy to measure the communication contention on the bus. Only one transaction can be performed on the bus at each time. The delay between the read request of the spy and the retrieval of the data is typically constant. However, in the presence of a communication collision, that is, if the victim is fetching one word from the shared L2 cache or the main memory, the spy will experience an increase in the data retrieval delay (4). Consequently, the spy is able to infer the victims access pattern when fetching the T-table stored in the shared L2 cache. After the 16th access (first round of AES), the spy can start the probing step. All lines in L2 that are accessed by the victim cause cache misses for the spy and subsequent requests to DRAM (5). All lines not accessed by the victim, cause immediate cache hits in L2 (6). After the probing step, the spy can start the key recovery phase. In case no T-tables are fetched by the victim from L2 (due to previous T-table accesses, thus keeping the T-tables in the core-private L1 cache), the probing step incorporates noisy measurements. However, the improvement of the attack compared to the original Prime+Probe technique remains.

B. Key Recovery

The analysis uses two types of information: i) the location of the cache sets where T-tables are stored (see Subsection V.C); and ii) the non-accessed sets. Based on this information, the spy is able to know which indexes have not been used by the victim. By exploiting Equation 1, the spy is able to retrieve key candidates in two steps. In the first step, the spy determines the

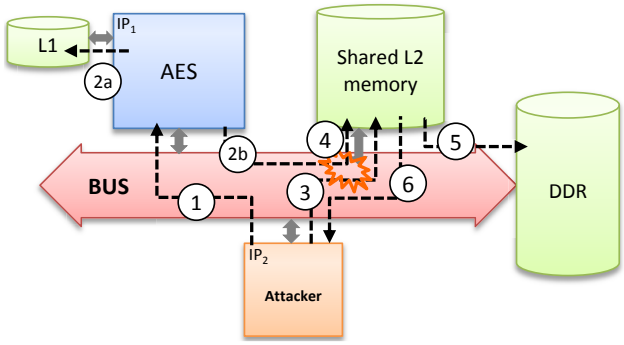

Fig. 3: Bus-based Optimized Prime+Probe Attack.

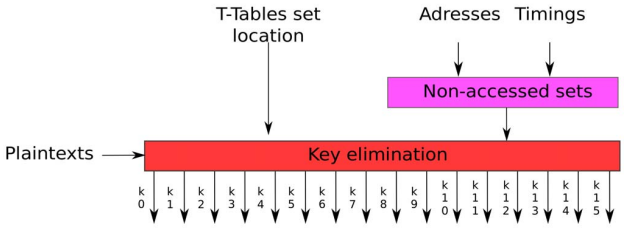

Fig. 4: Key elimination using cache observations.

non-accessed cache sets for table T_l. This yields a set of non-used indexes for T_l, denoted as I_{nu}. In the second step, the reduction of the key search space for K_i, K_{i+4}, K_{i+8}, K_{i+12} is performed. Possible values for (l, i) are $(0, 0)$, $(1, 1)$, $(2, 2)$ and $(3, 3)$. As a result, Equations (3) to (6) are obtained.

$$K_i = K_i \setminus (P_i \oplus I_{nu}) \tag{3}$$

$$K_{i+4} = K_{i+4} \setminus (P_{i+4} \oplus I_{nu}) \tag{4}$$

$$K_{i+8} = K_{i+8} \setminus (P_{i+8} \oplus I_{nu}) \tag{5}$$

$$K_{i+12} = K_{i+12} \setminus (P_{i+12} \oplus I_{nu}) \tag{6}$$

The two-step structure of the analysis is shown in Fig. 4. After parsing the memory addresses read by the spy during the probing phase and the corresponding timing information, the elimination of wrong key candidates is possible.

C. Challenges

In this work we assume that the L1 cache does not contain T-table elements before an encryption. This is true after booting the SoC and can also be enforced during runtime by three techniques: i) requesting the victim core to execute other code between encryptions, which leads to the self-eviction of T-table elements contained in the L1; and ii) waiting a certain period of time to ensure that all T-table entries have automatically been removed from the victim's L1 (due to the small size of the L1); iii) clearing the L1 by resetting the system.

The discovery of the T-table location in the shared L2 is essential to determine the non-accessed cache sets (note that the actual address of the T-table is not required). In the work by Osvik et al. [7], the spy performs the first round analysis by considering all possible T-table locations. The location that leads to the expected distinction of a set of key-byte hypotheses is considered to be the most probable location of the T-tables. In the work of Zhao et al. [14], the authors measure the access time variation for the cache sets after the execution of the Prime+Probe attack several times. Cache sets with higher access times during all the attacks are considered as the most probable T-table locations. In this work, we employ a similar technique to the one proposed in [7]. However, our analysis is based on non-accessed cache sets, which enhances the discovery process of the T-table cache sets. Wrong table offsets lead to empty sets of key candidates for at least one key byte. The first offset which leads to a non-empty set of key candidates is not necessarily the actual start address of T0, but an address that leads to the same cache set mapping.

VI. Attack Results and Countermeasures

This section presents the experimental setup, the evaluation of the attack and the impact of different L1 and L2 cache configurations. Finally, countermeasures and their evaluation are presented.

A. Experimental Setup

The bus-based SoC shown in Fig. 3 is implemented on the Nexys4 DDR FPGA with Vivado 2015.2 Design suite from Xilinx. It is composed of 6 IP cores: i) victim (Microblaze 9.5) running an AES T-table implementation; ii) private L1 cache; iii) shared L2 cache (system Cache 3.1); iv) DDR; v) spy module; and vi) AXI bus with round-robin arbitration. For expediting the repeatability and the execution of the attack, all IP cores used inside the SoC, except the spy module, are included in Vivado 2015.2. The spy module will be released and made publicly available in [20]. The spy is able to trigger an AES encryption on the Microblaze by means of an interrupt controller and the AXI timer. AXI4-Lite protocol is employed to perform the read/write transactions. The data gathered from the spy is transmitted to the PC through the UART interface (AXI Uartlite). Table I presents the characteristics of the memory retrieved from the specification.

B. Effectiveness of the Attack

Fig. 5 compares the effectiveness of the access-driven cache attack with the bus-based optimization (at the end of the first round) and the traditional attack (at the end of the encryption) after 100 plaintexts. Results show that our approach achieves a full key recovery (16 bytes) with 40 encryptions. The results show that our improved attack recovers the full key twice as fast as the Prime+Probe technique (without exploiting communication) presented in [14]. Moreover, our attack overcomes the previous NoC enhanced access-driven cache attack presented in [18], which requires 80 encryptions to recover only 12 key bytes and consequently introduces a brute-force search to determine the remaining 4 bytes (2^{32} possibilities). The high effectiveness of our attack is due to the discovery of an optimized probing point, i.e., at the end of the first round instead of the end of the encryption. Consequently, additional sources of noise due to T-tables accesses after the first round are eliminated. As the spy uses legitimate operations (memory read and write accesses to L2), firewall countermeasures are not feasible.

We evaluated the effectiveness of our attack due to the variation of the number of words per cache line in L1, (δ), and the sizes (α) of L1 and L2. Results show that an increase of δ decreases the number of L1 misses during the first round of AES. Since the attack exploits the L1 misses, a smaller δ will lead to a faster full key recovery. Changing $\delta = 4$ to $\delta = 8$ leads to an increase of up to 25% of the amount of traces required to retrieve the full key. Regarding the sizes of the caches, results show that small variations of α (2KB$< \alpha_{L1} <$ 8KB and 32KB$< \alpha_{L2} <$ 64KB) do not influence the attack. A deeper exploration of the configuration parameters of L1 and L2 is planned as a future work.

TABLE I: Details of the cache levels.

	Private L1 Cache	Shared L2 Cache
Configuration	Direct-mapped	4-Way Associative
Size	1 to 8 KB	32 to 128 KB
Words per line	4 to 8	16
Data Width	32 bits	32 bits
Write policy	Write-through	Write-back
Read hit	1 cycle	5 cycles
Read miss	2 cycles + LMem	2 cycles + LMem
Write hit	Variable	2 cycles
Write miss	Variable	2 cycles 6 cycles + LMem

Algorithm 1 Shuffle-Algorithm

```
1:  procedure SHUFFLE(permuted_sequence,LFSR,start)
2:
3:      values_taken [255:0]
4:      counter [8:0]
5:
6:      if start == 1 then
7:          values_taken ← (others → 0)
8:      else
9:          while values_taken ≠ (others ← 1) do
10:             i ← to_integer(LFSR[7 : 0])
11:             if values_taken(i)==0 and start==0 then
12:                 permuted_sequence(counter) ← i
13:                 counter ← counter + 1
14:                 values_taken(i) ← 1
15:             end if
16:         end while
17:     end if
18: end procedure
```

C. Countermeasures

This section presents two countermeasures: i) T-table Shuffling; and ii) Mini-table implementation. Shuffling is a hardware/software approach which modifies the T-table layout inside the shared L2 in order to avoid the localization of the

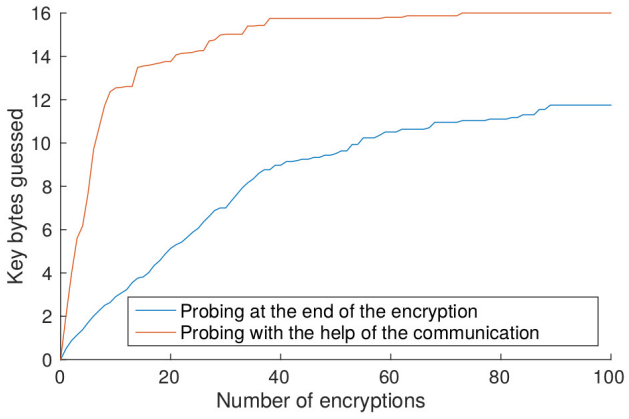

Fig. 5: Comparison of our approach (bus-based optimization) with the traditional approach (end of encryption).

TABLE II: Performance and cost overhead

Implementation	Perf. Impact	Additional Resources
T-table	1	-
Shuffling	4.1	LFSR
Mini-table	3.8	80 tables (of 64 bytes)

T-tables (see section V.C). A Linear Feedback Shift Register (LFSR) is used to generate pseudo-random numbers, which are processed by Algorithm 1. Using the LFSR, the algorithm computes five permuted T-tables. The second countermeasure, a Mini-table implementation, is a software approach which leads to a constant number of accessed shared cache lines. Thus, a spy will not be able to exploit the non-accessed cache lines. Mini-tables use a hierarchy of lookup tables that allow to uniformly spread the T-tables in the shared memory. Each T-table is split into n smaller T_i-tables, such that: i) each T_i-table contains d_i bits of each of the elements inside one T-table and fits exactly in one cache line; and ii) the computation of one table element $T[X]$ uses each of the T_i-tables as in (7).

$$T[X] = \sum_{i=0}^{n-1} T_i[X] * 2^{\sum_{j=0}^{i-1} d_j} \quad (7)$$

In our experiment, $\delta_{L2} = 16$ words (32 bits per line). Thus, a T_i-tables contains $d_i = \delta * 32/256 = 2bits$. Therefore, each line of L2 contains 2 bits of each T-table (T0-T4). The Mini-table implementation requires the definition of $16 * 5 = 80$ lookup tables. 16 cache lines are used to store one T-table. The protected implementation forces the access of all 16 lines for the computation of one lookup. Therefore, the cache traces contain 4*16 = 64 initial misses followed by hits for the lookup accesses of the first round (we assume that no T_i-tables are contained in the L1 before the encryption). To avoid first and last round based attacks on AES, these rounds are protected with Mini-tables. Remaining rounds use normal T-tables, leaving the speed optimized implementation for the intermediate rounds in place.

Table II shows the summary of the performance and additional resources overhead. After the integration of the countermeasures, the optimized access-driven cache attack becomes infeasible.

VII. Conclusion

In this work we propose an optimization of an access-driven cache attack by observing the communication in a bus-based SoC. This scenario is common in current SoCs, where peripherals and many other IP cores share a common bus. We show that by exploiting microarchitectural characteristics of the SoC, secret information of a cryptographic implementation is leaked. As a result, contrary to previous works, the full secret key can be retrieved by an adversary with only 40 encryptions. In order to protect the system, two mechanisms are presented: Shuffling and Mini-tables. We show that those countermeasures are feasible to be integrated into the SoC and are able to successfully thwart the presented attack. As future work, we aim to implement new protection mechanisms and further explore the effect of the memory hierarchy configuration on the effectiveness of the attack.

Acknowledgments. We thank the anonymous reviewers for their valuable comments and suggestions. This work was partly funded by the German Federal Ministry of Education and Research (BMBF), grant number 01IS160253 (ARAMiS II).

References

[1] "Accelerating the Data Plane With the TILE-Mx Many-core Processor," http://www.tilera.com/files/drim_EZchip_LinleyDataCenterConference_Feb2015_7671.pdf, accessed: 2017-03-13.
[2] "KALRAY MPPA: A New Era of processing," https://de.slideshare.net/\infokalray/kalray-sc13-external3, accessed: 2017-03-13.
[3] "Using Intels Single-Chip Cloud Computer (SCC)," https://communities.intel.com/docs/DOC-19269, accessed: 2017-03-13.
[4] D. J. Bernstein, "Cache timing attacks on aes," April 2005.
[5] A. Bogdanov, T. Eisenbarth, C. Paar, and M. Wienecke, "Differential cache-collision timing attacks on aes with applications to embedded cpus," in *The Cryptographers Track at the RSA Conference 2010, San Francisco, CA, USA, March 1-5, 2010. Proceedings*. Springer-Verlag, 2010, pp. 235–251.
[6] O. Acıiçmez and Çetin K. Koç, "Trace-driven cache attacks on aes," in *Proceedings of the 8th International Conference on Information and Communications Security*, ser. ICICS'06. Springer-Verlag, 2006, pp. 112–121.
[7] D. A. Osvik, A. Shamir, and E. Tromer, *Cache Attacks and Countermeasures: The Case of AES*. Springer Berlin Heidelberg, 2006.
[8] M. Lipp, D. Gruss, R. Spreitzer, C. Maurice, and S. Mangard, "ARMageddon: Cache Attacks on Mobile Devices," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, 2016, pp. 549–564. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/lipp
[9] X. Zhang, Y. Xiao, and Y. Zhang, "Return-oriented flush-reload side channels on arm and their implications for android devices," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: ACM, 2016, pp. 858–870.
[10] Z. H. Jiang, Y. Fei, and D. Kaeli, "A Complete Key Recovery Timing Attack on a GPU," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, March 2016, pp. 394–405.
[11] C. Reinbrecht, A. Susin, L. Bossuet, and J. Sepulveda, "Gossip NoC - Avoiding Timing Side-Channel Attacks through Traffic Management," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI '16)*, July 2016.
[12] R. Fernandes, R. Cataldo, C. Marcon, G. Sigl, and J. Sepúlveda, "A Security aware Routing Approach for NoC-Based MPSoC," in *Integrated Circuits and Systems Design (SBCCI), 2016 29th Symposium on*. IEEE, 2016, pp. 1–6.
[13] M. Neve and J.-P. Seifert, "Advances on access-driven cache attacks on aes," in *Selected Areas in Cryptography*, ser. Lecture Notes in Computer Science, E. Biham and A. Youssef, Eds. Springer Berlin Heidelberg, 2007, vol. 4356, pp. 147–162.
[14] X. Zhao, W. Tao, M. Dong, Z. Yuanyuan, and L. Zhaoyang, "Robust first two rounds access driven cache timing attack on aes," in *Computer Science and Software Engineering, 2008 International Conference on*, vol. 3, Dec 2008, pp. 785–788.
[15] G. Irazoqui, T. Eisenbarth, and B. Sunar, "S$a: A shared cache attack that works across cores and defies vm sandboxingand its application to aes," in *Security and Privacy (SP), 2013 IEEE Symposium on*, 2015.
[16] R. Spreitzer and T. Plos, "Cache-access pattern attack on disaligned aes t-tables," in *Constructive Side-Channel Analysis and Secure Design*, ser. Lecture Notes in Computer Science, E. Prouff, Ed. Springer Berlin Heidelberg, 2013, vol. 7864, pp. 200–214.
[17] M. J. Sepúlveda, J.-P. Diguet, M. Strum, and G. Gogniat, "NoC-Based Protection for SoC Time-Driven Attacks," *IEEE Embedded Systems Letters*, vol. 7, no. 1, pp. 7–10, 2015.
[18] C. Reinbrecht, A. Susin, L. Bossuet, G. Sigl, and J. Seplveda, "Side-Channel Attack on NoC-based MPSoCs are Practical: NoC Prime+Probe Attack," in *2016 29th Symposium on Integrated Circuits and Systems Design (SBCCI)*, Aug 2016, pp. 1–6.
[19] J. Daemen and V. Rijmen, *The Design of Rijndael*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2002.
[20] "Mpsoc attacks," https://www.sec.ei.tum.de/, accessed: 2017-05-06.