Earthquake - A NoC-based Optimized Differential Cache-Collision Attack for MPSoCs

Cezar Reinbrecht, Bruno Forlin
Instituto de Informática - PPGC - UFRGS
Porto Alegre, Brazil
cezar.reinbrecht@inf.ufrgs.br

Andreas Zankl
Fraunhofer AISEC
Munich, Germany
andreas.zankl@aisec.fraunhofer.de

Johanna Sepúlveda
Technical University of Munich
Munich, Germany
johanna.sepulveda@tum.de

Abstract—**Multi-Processor Systems-on-Chips (MPSoCs) are a platform for a wide variety of applications and use-cases. The high on-chip connectivity, the programming flexibility, and the reuse of IPs, however, also introduce security concerns. Problems arise when applications with different trust and protection levels share resources of the MPSoC, such as processing units, cache memories and the Network-on-Chip (NoC) communication structure. If a program gets compromised, an adversary can observe the use of these resources and infer (potentially secret) information from other applications. In this work, we explore the cache-based attack by Bogdanov et al., which infers the cache activity of a target program through timing measurements and exploits collisions that occur when the same cache location is accessed for different program inputs. We implement this differential cache-collision attack on the MPSoC Glass and introduce an optimized variant of it, the Earthquake Attack, which leverages the NoC-based communication to increase attack efficiency. Our results show that Earthquake performs well under different cache line and MPSoC configurations, illustrating that cache-collision attacks are considerable threats on MPSoCs.**

Index Terms—**Network-on-Chip, Security NoC, Timing Attack, Timing Side-channel Attack.**

I. Introduction

System-on-Chip (SoC) platforms have established themselves as cost-effective solutions for end consumer markets and industry domains alike. Rising demands for increased performance and lower energy consumption are nowadays driving new platforms, so-called Multi-Processor Systems-on-Chips (MPSoCs). These platforms are composed of several processing elements, custom IP cores, and memories, all of which are in general interconnected by a Network-on-Chip (NoC). MPSoCs drive application versatility and thrive on connectivity similar to Cloud Computing and the Internet-of-Things (IoT). Unfortunately, high versatility and connectivity introduce new threats to MPSoCs, as multiple applications with different trust and origin are sharing these systems and their resources.

One of the potent threats that MPSoCs see themselves exposed to are so-called side-channel attacks (SCA). This category of attacks relies on the inference of potentially secret information, such as personal data, passwords, or cryptographic keys, from observations of the system that processes them. These observations can target the physical behavior of devices, e.g., their thermal [1], power [2], [3], [4], [5], or electromagnetic [6] properties. This work, in contrast, focuses on timing leakage, i.e., the fact that different (secret) inputs take different amounts of time to be processed. In early literature, Kocher [7] initially described how the execution time of a cryptographic operation allows to infer the secret key involved in the computations.

In the multi-processor domain, the wide-spread use of cache memories has made timing attacks a very immanent threat. This is because the access speed of a processor to cache memory is often an order of magnitude higher than to the main memory. As a consequence, cache-based attacks routinely exploit different access times within the memory hierarchy. Unfortunately, design choices of cryptographic implementations have even facilitated these kind of attacks. One prominent and thoroughly studied example is the Advanced Encryption Standard (AES) and its software implementations that frequently rely on large look-up tables. Although they significantly improve speed, the secret-dependent accesses to these tables have repeatedly been exploited in cache attacks.

In general, cache attacks can be categorized in *trace-driven*, *access-driven*, and *time-driven* attacks. For trace-driven approaches, the adversary must observe the sequence and results of memory requests to the cache hierarchy. The trace of cache hits and misses gives insight into the data that has been processed [8]. For access-driven attacks, the adversary actively interferes with the cache activity of the target and thereby learns which data or code has been used during an operation [9]. Time-driven attacks, on the contrary, simply rely on the fact that cache hits and misses cause measurable differences in the overall processing time. Multiple observations of the processing time for different inputs then allow to draw a statistical conclusion about the data and code used during an operation [10], [11], [12].

In 2010, Bogdanov et al. [13] proposed a time-driven attack that exploits so-called cache collisions. They occur when intermediate values during an operation are identical for different inputs and subsequently cause accesses to the same or to a similar memory location. These accesses are then said to *collide*. Naturally, the first access will bring the requested memory into the cache hierarchy. This speeds up the second access and consequently the overall execution. For AES software implementations, Bogdanov et al. showed that different inputs can cause up to five cache collisions during encryption or decryption. This so-called *wide-collision* is sufficient to infer the secret key from a set of timing observations. To execute the attack, an adversary must be able to choose the input that is processed by an AES implementation and measure its execution time.

In this work, we show how the attack by Bogdanov et al. can be applied to an MPSoC setting and analyze its practicality in this novel scenario. Subsequently, we combine the attack with established methods from a previous NoC timing attack by Reinbrecht et al. [14] and show that the resulting approach, the *Earthquake* attack, is significantly more potent. Our results emphasize once more that timing attacks, in particular those exploiting the cache hierarchy, are an immanent threat on NoC-based MPSoCs and must be addressed during design time to ensure the security of the overall system. In summary, our contributions are

- the implementation and evaluation of a differential cache-collision attack on a real MPSoC hosted on an FPGA,
- the analysis of the practicability of the attack in a multi-core environment,
- the proposal of a new cache-based attack, the Earthquake attack, which successfully combines cache-collisions with techniques from NoC timing attacks.

The rest of the paper is organized as follows. Section II provides background information about the AES and our reference MPSoC. Section III presents previous works related to cache collision attacks. In Section IV, the differential cache-collision attack by Bogdanov et al. is explained in detail. Section V presents the proposed Earthquake attack. Section VI shows the experimental results of both attacks running on an FPGA, before we finally conclude in Section VII.

II. Background

This section briefly introduces the Advanced Encryption Standard (AES), its relation to side-channel attacks, and the reference multi-processor system-on-chip (MPSoC) we use in our experiments.

A. Advanced Encryption Standard (AES)

AES is a widespread symmetric cipher that operates on inputs of 128 bits and uses keys of 128, 192 or 256 bits. Encryption and decryption are implemented in rounds that each consists of four elementary operations. The longer the key, the more rounds are executed, e.g. 10 rounds for a 128-bit key and 14 rounds for a 256-bit key. The input to AES is organized in blocks of 16 bytes. For encryption, the plaintext is written as p_i, where $0 \leq i \leq 15$. Each input block is arranged as a 4x4 state matrix. The same holds for a 128-bit key, which is written as k_i. As each round uses a distinct round key, a so-called key expansion is performed. This yields k_i^r, where $0 \leq r \leq 9$. The previous k_i is the initial key in round 0, i.e., k_i^0. For encryption, the four elementary operations are as follows:

- *SubBytes*: substitute all input bytes with lookups from the so-called substitution box, or S-box
- *ShiftRows*: shift each row of the state matrix cyclically to the left, the row index represents the shift value
- *MixColumns*: polynomial multiplication over $GF(2^8)$, each column represents a set of coefficients and is replaced by the multiplication result
- *AddRoundKey*: XOR operation between the state matrix and the current round key

In the last round of AES, the MixColumns operation is omitted. For decryption, the inverse round operations are used and performed in reverse order.

Performance-oriented Implementation: To improve the performance of AES implementations, the round transformations SubBytes, ShiftRows, and MixColumns can be performed using tables of pre-computed values, so-called *transformation* or *T-tables* [15]. This introduces four 1 kB tables (T_0, T_1, T_2, and T_3) as well as one additional table T_4 for the last round, which omits the MixColumns transformation. The AddRoundKey operation remains unchanged.

Implementations using T-tables reduce each round to two operations. First, the state matrix, represented as x_i^r, where $0 \leq i \leq 15$ and $0 \leq r \leq 9$, is used as input to the T-tables. The result of the lookups is XOR'ed to form an updated state matrix. Second, the AddRoundKey operation is performed to update the state with the current round key. Equation 1 illustrates this reduced round of AES.

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

The last round is computed by substituting $T_0, ..., T_3$ with T_4. The resulting x_i^{10} is the final ciphertext.

B. Multi-Processor System-on-Chip

Market demands like high performance and low energy consumption have driven the development of versatile platforms known as Multi-Processor System-on-Chips. MPSoCs are complete systems containing multiple processing elements on the same integrated circuit [16]. Besides the processors, the system can comprise co-processors, hardware accelerators, memories and a Network-on-Chip (NoC). NoCs are a typical interconnection solution to integrate several components and can vary in router architecture or topology. Applications and fields in which MPSoCs are already used include Machine Learning, Internet-of-Things, high-bandwidth communication, augmented reality, and video encoding/decoding. Any MPSoC can be classified according to the following characteristics:

- the architecture model: which processors and IPs comprise the system.
- the memory model: shared, distributed or distributed-shared strategy.
- the communication model: bus-based or NoC-based topology.
- the software architecture: hardware abstraction layer (HAL), operating system (OS), application programming interfaces (APIs).

C. Reference Architecture - MPSoC Glass

The reference architecture is the MPSoC Glass, a heterogeneous MPSoC interconnected by a mesh NoC using a shared memory organization. The structure and components can be observed in Figure 1. MPSoC Glass uses the NIOS II processor from Altera. Similar to ARM's big.LITTLE technology [17], our architecture uses different NIOS II implementations, the economy core (NIOS II/e from [18]), and the fast core (NIOS II/f from [18]). The NIOS II fast core aims at high-performance applications, system management and primary tasks of the system. Minor tasks and parallel applications are intended to be executed by a group of NIOS II economy cores. The NIOS II economy core has a low area and power, and is suitable to be replicated many times in an MPSoC. Consequently, there are more economy cores in the MPSoC than fast cores, i.e., four fast NIOS II and ten economy NIOS II. Other components integrated into the MPSoC Glass are a shared cache, and an UART interface resulting in a total of sixteen parts.

The memory hierarchy has two cache levels. The level 1 (L1) is a local instruction and data direct-mapped cache. The level 2 (L2) is a shared unified cache and 16-way set-associative. The shared L2 cache has a direct link to the

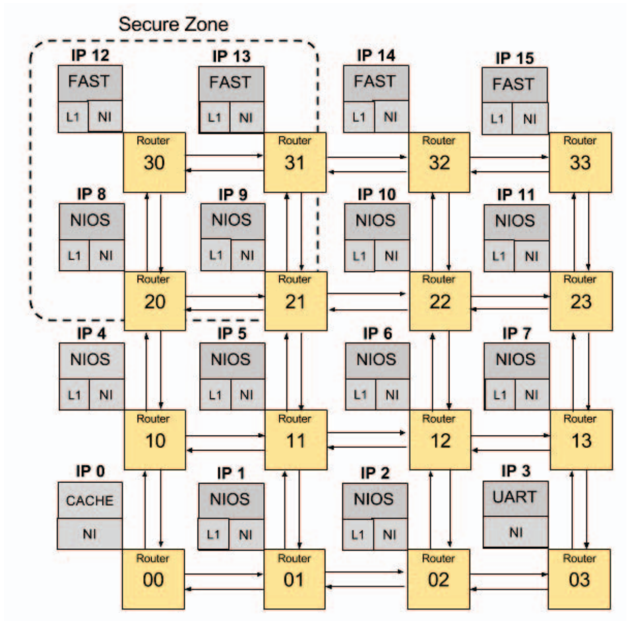

Fig. 1: MPSoC Glass: Four fast NIOS II, ten economy NIOS II, one UART interface and one shared cache memory.

external main memory. The sizes of the caches and cache lines are parametrized for each design. Depending on the target usage, a different configuration can be set.

The NoC is composed of a 5-port router, with input buffers of eight flits, each flit having 32 bits. The topology is a 4x4 mesh, which uses the XY routing algorithm. Also, MPSoC Glass has a built-in security zone, defined at design-time. Elements inside secure zones mutually trust each other, because their communication is exclusive [19] [20]. The processors inside the secure zone are not allowed to download or run untrusted software. Processors outside the secure zone can execute any application, even external ones downloaded by the system.

The software architecture is simplified for the first version of MPSoC Glass. Each processor has a hardware abstraction layer (HAL) to initialize the necessary components, and a bare-metal code that runs the tasks directly in the HAL. A developed library set provides all communication features through the NoC and cryptographic functions as services, being similar to an API solution. A custom development environment was used to automate parts of the development flow, such as compilation and upload of the binaries in the several processors of the system.

III. Related Work

In literature, cache-collision attacks on table-based implementations of AES have been proposed by Acıiçmez et al. [21], Bonneau and Mironov [11], and Bogdanov et al. [13]. The key concept behind the attacks is that identical intermediate values in the first or last rounds of AES cause lookups to identical entries of the T-tables. In general, this is called an internal collision. Since multiple T-table entries fit on one cache line, even similar intermediate values can cause collisions in the cache, i.e., they use the same cache line for the lookup. On average, cache collisions cause a reduction in the overall execution time of AES. The inputs causing the cache collisions are then used to infer the secret key.

The attack proposed by Acıiçmez et al. [21] exploits internal collisions in the first two rounds of AES. It is a chosen-plaintext attack and allows the authors to recover a full AES key with $2^{26.66}$ queries when timing the communication via the operating system's network stack. Bonneau and Mironov [11] describe multiple cache-collision attacks targeting the first and last round of AES. While the first round attack only allows to retrieve a limited number of key bits, the last round attacks allow full-key recovery. The authors also explicitly leverage cache collisions caused by similar, but not identical, intermediate values. With this improvement, the authors are able to recover a full key with less than 2^{19} encryptions.

Bogdanov et al. [13] generalize the idea of exploiting cache collisions in AES. The authors establish the notion of *wide-collisions*, which are five internal collisions in the first three rounds of AES. Wide-collisions do not exclusively apply to T-table based AES implementations and are thus a generic attack vector. In T-table implementations, they manifest themselves as cache collisions and consequently reduce the runtime of AES. Wide-collisions are triggered with chosen plaintexts and detected by timing the encryption. With this attack, the authors are able to reduce the brute-force complexity of AES-128 to $2^{36.9}$ using $2^{27.97}$ encryptions.

Spritzer and Plos [22] investigate the applicability of Bogdanov's attack on Android-based mobile devices. The authors show that the reliable detection of wide-collisions is challenging in practice and that large, i.e. 64-byte, cache lines further decrease the success rate. The authors conclude that no practical attack is possible in the chosen attack scenario.

Lauradoux [23] demonstrates that cache collisions cannot not only be detected through timing measurements, but also through measurements of the power consumption. The author argues that cache misses have a different power profile than cache hits and that it is therefore possible to also detect cache collisions. In this work, we investigate only timing measurements, as the power consumption can typically not be determined without additional equipment and physical access to the target device.

IV. Differential Cache-Collision Attack

The work by Bogdanov et al. [13] describes a chosen-plaintext attack that exploits wide-collisions in software implementations of AES. To start the attack, the adversary generates plaintext pairs (P_1 and P_2) that are consecutively encrypted by the target implementation. Each pair targets one of four diagonals found in the 4x4 state matrix of AES. The pairs are constructed such that the values of the bytes in the target diagonal are different between the pairs, whereas all other bytes are identical. The example below shows a plaintext pair that targets the main diagonal. The diagonal elements in P_1 (labeled as a_i) are different from those in P_2 (labeled as e_i). All elements x_j are identical in both plaintexts. The concrete values of all a, e, and x are chosen randomly.

$$P_1 = \begin{bmatrix} a_0 & x_1 & x_2 & x_3 \\ x_4 & a_5 & x_6 & x_7 \\ x_8 & x_9 & a_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & a_{15} \end{bmatrix} \qquad P_2 = \begin{bmatrix} e_0 & x_1 & x_2 & x_3 \\ x_4 & e_5 & x_6 & x_7 \\ x_8 & x_9 & e_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & e_{15} \end{bmatrix}$$

After computing the first round of AES, the state matrices of both plaintexts at the beginning of the second round (P_1^2 and P_2^2) look as follows. The matrix elements that correspond to a_i and e_i are highlighted in gray.

$$P_1^2 = \begin{bmatrix} s_0 & s_4 & s_8 & s_{12} \\ s_1 & s_5 & s_9 & s_{13} \\ s_2 & s_6 & s_{10} & s_{14} \\ s_3 & s_7 & s_{11} & s_{15} \end{bmatrix} \qquad P_2^2 = \begin{bmatrix} s_0 & s_4 & s_8 & s_{12} \\ s_1 & s_5 & s_9 & s_{13} \\ s_2 & s_6 & s_{10} & s_{14} \\ s_3 & s_7 & s_{11} & s_{15} \end{bmatrix}$$

The original plaintext pair causes a wide-collision, if at least one collision happens at the input of the S-box in the second round. In the illustrated example, at least one of the highlighted s_i must be identical for both P_1^2 and P_2^2. This identical byte then causes four more collisions in the S-box lookups of the third round. Assuming that s_0 collides in the example (highlighted in green), the following matrices illustrate the input of the third round, P_1^3 and P_2^3:

$$P_1^3 = \begin{bmatrix} t_0 & t_4 & t_8 & t_{12} \\ t_1 & t_5 & t_9 & t_{13} \\ t_2 & t_6 & t_{10} & t_{14} \\ t_3 & t_7 & t_{11} & t_{15} \end{bmatrix} \qquad P_2^3 = \begin{bmatrix} t_0 & t_4 & t_8 & t_{12} \\ t_1 & t_5 & t_9 & t_{13} \\ t_2 & t_6 & t_{10} & t_{14} \\ t_3 & t_7 & t_{11} & t_{15} \end{bmatrix}$$

As shown, the first columns of both state matrices are identical, which causes four more identical S-box lookups. In total, this yields five internal collisions, or a wide-collision. The following paragraphs outline how wide-collisions are obtained and leveraged to recover the secret AES key from timing measurements.

Online Stage: The main goal of the online stage is to generate plaintext pairs and measure the encryption time of all P_2 plaintexts. For this, N unique plaintext pairs with different diagonal bytes (a_i, e_i) are chosen. For each of these pairs, the other bytes (x_j) are changed I times. These $N \cdot I$ plaintext pairs are then encrypted R times each. The parameter N thereby defines the number of candidates that might cause a wide-collision. The parameter I improves the detection of a wide-collision and rules out particularly unfortunate choices of x_j. The parameter R simply reduces the noise of the timing measurements. The output of the online stage are the encryption times of all processed P_2 plaintexts.

Detection Stage: In the detection stage, each of the N unique pairs is assigned a final timing value, which is the minimum encryption time over all corresponding I executions. The 4 pairs with the lowest timings are then classified as wide-collision candidates. In order to recover the entire key with only 2^{32} search complexity, there must be at least 4 true wide-collisions among the chosen candidates. In general, if m candidates are chosen, the final search complexity increases to $\binom{4+m}{4} \cdot 2^{32}$.

Key Recovery Stage: During the key recovery stage, the previously chosen plaintext pairs are used to prune the list of possible key choices. Since the candidates of each diagonal contain information about 4 bytes of the full key, every candidate list is used to prune the corresponding 2^{32} choices. Each sub-key choice is thereby tested against the candidate list. If a wide-collision would occur for the given list, the sub-key choice is kept.

All stages must be performed for each of the four diagonals. The remaining sub-key choices are then combined and exhaustively searched for the correct key.

V. Earthquake Attack

The differential cache-collision attack by Bogdanov et al. [13] targets internal collisions that occur during the first three rounds of AES. These collisions influence the overall execution time, but are not the only source of timing variations. The remaining rounds add noise to the timings and thereby increase the number of measurements required for a successful attack. Consequently, if an adversary is able to measure the execution time of only the first three rounds, the attack is expected to succeed with significantly fewer observations. This is the key improvement of the Earthquake attack compared to the original approach. The following paragraphs discuss how this improvement can be achieved on NoC-based MPSoCs and how the Earthquake attack can be launched in such a setting.

Attack Scenario: The target platform of the Earthquake attack is any MPSoC that implements a network-on-chip. As modern MPSoCs can host numerous components that are constantly growing in complexity, it is reasonable to assume that one of them can eventually be compromised by an adversary [24]. To start the Earthquake attack, this compromised component must be able to trigger AES encryptions with chosen inputs, either directly or indirectly. The AES functionality might for instance be offered by a cryptography API or by a centralized service manager running on one of the MPSoC components. Given that the AES implementation uses T-tables, the adversary component must also be located on the communication path between the AES service and the shared cache level of the MPSoC. Adversaries located on the sensitive path are able to sense the communication pattern and volume. The goal is to annotate the execution time at the end of the third round (after 48 T-table accesses).

Cache hierarchy and configuration affect the attack. The L1 cache is typically a local memory located in the processor tile. Communication between the processor and the L1 cache will bypass the NoC and therefore cannot be observed by the attacker. This makes detecting the end of the third AES round more difficult and introduces noise in the timing measurements. In contrast, the communication between the processor and the L2 cache is observable by the attacker. Although the local L1 cache has an adverse effect on timing measurements, its impact was limited in our experiments and did not require further action.

Attack Steps: The Earthquake attack consists of nine steps that are illustrated in Figure 2. The first step of the attack is the *infection* of MPSoC components. The more components are compromised, the more likely it is for an attacker to be located on the sensitive path between the L2 cache and the AES component. The next three steps include choosing a target diagonal, generating plaintext pairs, and choosing multiple sets of x_j per plaintext pair.

The subsequent step performs the measurement phase, which in turn consists of four sub-steps. They are highlighted in Figure 2 in a separated box with dashed lines. In the *prime* step, the shared cache is filled with random data from the adversary. This ensures that no T-table entries reside in the cache before the first encryption. In the subsequent step, the adversary triggers the encryption of one plaintext pair. During the second encryption, the adversary uses the approach proposed by Reinbrecht et al. [14] to observe the end of the third round. In their work, the authors show that an adversary can observe accesses to a shared cache by saturating the

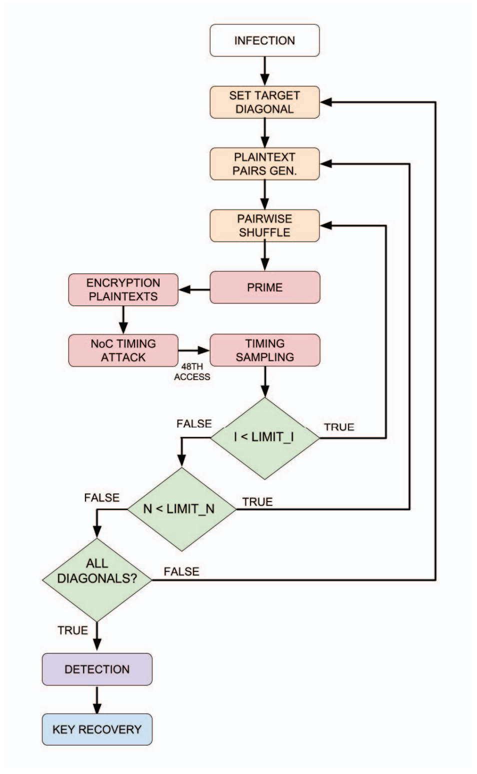

Fig. 2: Earthquake attack steps. The measurement parameter R is set to 1 and is therefore not considered further in the attack steps.

communication path to the cache with own traffic. Accesses by other components to the cache naturally slow down own accesses. This can be detected and allows the adversary to infer other cache activity. In the Earthquake attack, this technique is used to measure the time until the AES service has made 48 accesses to the cache. This is the number of T-table accesses during the first three rounds. In this way, the adversary can accurately detect the end of the third round and thereby significantly improve the detection of wide-collisions.

After the measurement step, the *detection* and *key recovery* steps are performed as proposed in the original attack. In general, these final steps need not be performed on the MPSoC. Instead, an adversary can transmit the measurement results to an external device that offers more computational power, which speeds up the key recovery phase.

A. NoC Timing Attack In Earthquake

Figure 1 shows the attack scenario used in our experiments. A sensitive process (S) on IP_{13} is offering a performance-oriented AES encryption function (T-table implementation). The process uses four lookup tables (T_0, T_1, T_2, and T_3) to perform the SubBytes, ShiftRows, and MixColumns transformations. When an encryption is requested, the AES implementation accesses the T-tables, thus, issuing data requests to the private $L1_{13}$ cache. Next, the availability of the data in $L1_{13}$ is checked. If the data is not in L1, a miss occurs and an access to $L2_0$ is performed. This generates sensitive communication through the NoC from IP_{13} to IP_0. The traffic must follow the deterministic path (sensitive path)

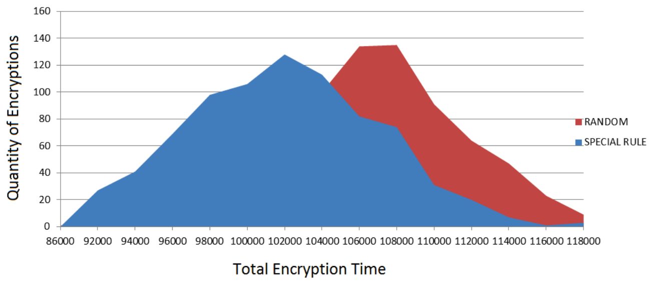

Fig. 3: Histogram of encryption times for pairs of random plaintexts and pairs used in Bogdanov et al.'s attack.

determined by the XY routing algorithm, which includes five routers ($Router_{31}, Router_{30}, Router_{20}, Router_{10}$ and $Router_{00}$). The response message from IP_0 to IP_{13} uses another five routers ($Router_{00}, Router_{01}, Router_{11}, Router_{21}$ and $Router_{31}$).

Simultaneously, a malicious process (M) is running on the infected IP_1, which is located on the sensitive path between the L2 cache and the AES process ($Router_{01}$). The infected IP_1 injects frequent dummy transactions to saturate a particular output port (North Port) of the $Router_{01}$. Due to the shared nature of the router resources, this causes communication collisions in the North output port between malicious and sensitive traffic. M annotates its own throughput by using the time of successful packet injection in the router. M's throughput then reveals the access pattern and communication volume of the sensitive flow, as low throughput shows the existence of collisions inside the router. The security zone shown in Figure 1 does not prevent the attack in this scenario, as the communication is observed outside of it. Placing the L2 cache within the security zone reduces the attack surface, but limits accessibility to the shared cache for non-secure components.

VI. Experimental Results and Analysis

In order to compare the original attack proposed by Bogdanov et al. [13] and the improved Earthquake attack, both of them are implemented on the MPSoC Glass shown in Figure 1. Each of the NIOS II cores feature a 32 kB private L1 cache. The shared L2 cache is 256 kB in size and it is inclusive of L1. All of them have a cache line size of 16 bytes.

The first experiment that was conducted demonstrates the general practicality of our approach. In this experiment, two sets of plaintext pairs were generated. The first set contains purely random plaintexts. The second set contains pairs created according to Bogdanov et al.'s attack. Then, the execution time of the first three rounds was measured for all pairs in both sets. The resulting timing histograms are shown in Figure 3. The random plaintext set is colored in red, the crafted plaintext set is colored in blue. As expected, the plaintexts used in the attack invoke more wide-collisions as the random plaintexts. This clearly demonstrates that our approach of measuring the end of the third round is feasible in practice and yields exploitable timing differences.

A. Attack Comparison

In order to fairly evaluate and compare both attacks, the measurement parameters N, representing the number of different values for the target diagonal, and I, representing the

TABLE I: False-positive wide-collisions per diagonal among the ten fastest plaintext pairs.

Attack Parameters		False-Positives		Earthquake
N	I	Bogdanov	Earthquake	Improvement
50	200	1.25	0.75	40%
	400	1.25	0.25	80%
	600	1.00	0.50	50%
100	200	1.00	0.75	25%
	400	1.00	0.25	75%
	600	1.00	0.25	75%
150	200	0.75	0.75	0%
	400	0.75	0.00	100%
	600	0.50	0.25	50%
250	200	1.00	0.25	75%
	400	0.50	0.00	100%
	600	0.50	0.00	100%

number of different pairs for every diagonal value, are varied as follows: $N_1 = 50$, $N_2 = 100$, $N_3 = 150$ and $N_4 = 250$; $I_1 = 200$, $I_2 = 400$ and $I_3 = 600$. The parameter R, which is used to reduce measurement noise, is set to 1 in our experiments. This is because NoC environments are less noisy than local or wide area networks such as the Internet.

To evaluate the effectiveness of both attacks, measurements are taken for all combinations of the measurement parameters. From the obtained timings, the ten most likely wide-collision candidates are selected and tested for correctness. The corresponding numbers of false-positives are shown in Table I for both attacks. False-positives significantly pro-long or break the exhaustive key search phase. Therefore, the lower the number of false-positives, the higher the success probability of the attack.

The results in Table I show that increasing N or I generally improves the attacks. Larger values for I improve the detection of wide-collisions that are caused by a given diagonal. However, increasing I also prolongs the execution time of the attack. In our experiments, the MPSoC operates at 50MHz. With $I = 200$, the attack takes roughly eight hours to finish. By increasing I to 600, the attack took approximately twenty-five hours. Compared to Bogdanov et al.'s attack, the Earthquake attack is successful with fewer measurements (lower N and I). This renders the attack more practical. Table I shows that for $N = 250$ and $I = 200$, Earthquake produced only 0.25 false-positives per diagonal, whereas the original attack produced one false-positive per diagonal.

Compared to the results provided in the original work [13], the Earthquake attack performs significantly better. Bogdanov et al. achieve a remaining attack complexity of $2^{36.9}$ with $2^{27.97}$ encryptions ($N = 1024$, $I = 800$, $R = 40$). In contrast, Earthquake achieves a remaining attack complexity of $2^{32.0}$ with $2^{20.19}$ encryptions ($N = 250$, $I = 600$, $R = 1$). The number of encryptions is also lower than the $2^{26.66}$ queries consisting of 1024 encryptions each that Acıiçmez et al. [21] required for full key recovery. These results clearly illustrate that cache-collision attacks, and in particular the Earthquake attack, are a considerable threat in MPSoC settings.

VII. Conclusion

In this paper, we presented Earthquake, a novel cache-collision attack that further explores the vulnerabilities identified in Bogdanov et al.'s work. Earthquake allows to measure only the first three rounds of AES, which significantly improves the detection of wide-collisions. It also reduces the number of measurements and therefore the overall attack time. This is possible by jointly exploiting the shared cache level and the resources in the NoC communication infrastructure. The improvements turn the attack by Bogdanov et al. into a practical threat on modern MPSoC platforms. With our results we hope to bring attention to the danger these attacks represent, as well as to the importance of defense strategies focused on securing the cache hierarchy and the NoC communication infrastructure.

Acknowledgments. This work was partly funded by the German Federal Ministry of Education and Research (BMBF), grant number 01IS160253 (ARAMiS II).

References

[1] M. Hutter and J.-M. Schmidt, "The temperature side channel and heating fault attacks," in *CARDIS 2013*, Berlin, Germany, 2014, pp. 219–235.

[2] P. C. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *CRYPTO '99*, London, UK, UK, 1999, pp. 388–397.

[3] C. H. Gebotys and R. J. Gebotys, "Secure elliptic curve implementations: An analysis of resistance to power-attacks in a dsp processor," in *CHES 02*, London, UK, UK, 2003, pp. 114–128.

[4] M. e. a. Masoomi, "A practical differential power analysis attack against an fpga implementation of aes cryptosystem," in *Information Society (i-Society), 2010 International Conference on*, London, UK, UK, June 2010, pp. 308–312.

[5] S. e. a. Ors, "Power-analysis attack on an asic aes implementation," in *ITCC 2004.*, Las Vegas, NV, USA, USA, April 2004, pp. 546–552.

[6] K. e. a. Gandolfi, "Electromagnetic analysis: Concrete results," in *CHES 2001*, Berlin, Heidelberg, 2001, pp. 251–261.

[7] P. C. Kocher, "Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems," in *CRYPTO 96*, London, UK, UK, 1996, pp. 104–113.

[8] 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.

[9] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: The case of aes." Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 1–20.

[10] D. J. Bernstein, "Cache timing attacks on aes," Available at: https://cr.yp.to/antiforgery/cachetiming-20050414.pdf, April 2005, accessed at 2016-12-31.

[11] J. Bonneau and I. Mironov, "Cache-collision timing attacks against aes," in *Cryptographic Hardware and Embedded Systems - CHES 2006*. Yokohama, Japan: Springer Berlin Heidelberg, October 2006, pp. 201–215.

[12] M. e. a. Neve, "A refined look at bernsteinś aes side-channel analysis," in *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security*, New York, NY, USA, 2006, pp. 369–369.

[13] A. e. a. Bogdanov, "Differential cache-collision timing attacks on aes with applications to embedded cpus," in *CT-RSA 2010*, Berlin, Heidelberg, 2010, pp. 235–251.

[14] C. e. a. Reinbrecht, "Gossip noc - avoiding timing side-channel attacks through traffic management," in *ISVLSI 16*, Pittsburgh, USA, 2016, pp. 601–606.

[15] J. Daemen and V. Rijmen, *The Design of Rijndael*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2002.

[16] W. Wolf, A. A. Jerraya, and G. Martin, "Multiprocessor system-on-chip (mpsoc) technology," *IEEE TCAD*, vol. 27, no. 10, pp. 1701–1713, Oct 2008.

[17] ARM, "White paper - big.little technology: The future of mobile," Available at: https://www.arm.com, note = Accessed at 2017-10-07,.

[18] I. F. Altera, "Nios ii classic processor reference guide," Available at: https://www.altera.com/, note = Accessed at 2017-10-07,.

[19] J. e. a. Sepulveda, "Efficient and flexible noc-based group communication for secure mpsocs," in *ReConFig 2015*, Mexico City, Mexico, 2015, pp. 1–6.

[20] J. Sepulveda, "Elastic security zones for noc-based 3d-mpsocs," in *ICECS 2014*, Marseille, France, 2014, pp. 506–509.

[21] O. Acıiçmez, W. Schindler, and Çetin K. Koç, "Cache based remote timing attack on the aes," in *CT-RSA 2007*, 2006, pp. 271–286.

[22] R. Spreitzer and T. Plos, "On the applicability of time-driven cache attacks on mobile devices (extended version)," pp. 656–662, 2013.

[23] C. Lauradoux, "Collision attacks on processors with cache and countermeasures," in *WEWoRC'05*, 2005, pp. 76–85.

[24] L. e. a. Fiorin, "A security monitoring service for nocs," in *CODES+ISSS 08*, New York, NY, USA, 2008, pp. 197–202.