

SideLine: How Delay-Lines (May) Leak Secrets from your SoC

Joseph Gravelier¹, Jean-Max Dutertre², Yannick Teglia¹, and Philippe Loubet Moundi¹

¹ Thales, La Ciotat, France

{name.surname}@thalesgroup.com

² Mines Saint-Etienne, CEA-Tech, Centre CMP. Gardanne, France

dutertre@emse.fr

Abstract. To meet the ever-growing need for performance in silicon devices, SoC providers have been increasingly relying on software-hardware cooperation. By controlling hardware resources such as power or clock management from the software, developers earn the possibility to build more flexible and power efficient applications. Despite the benefits, these hardware components are now exposed to software code and can potentially be misused as open-doors to new kind of attacks. In this work, we introduce *SideLine*, a novel side-channel vector based on delay-line components widely implemented in high-end SoCs. We demonstrate that these entities can be used to perform remote power side-channel attacks and we detail several attack scenarios in which an adversary process located in one processor core aims at eavesdropping the activity of a victim process located in another core. For each scenario, we demonstrate the adversary ability to fully recover the secret key of an AES algorithm running in the victim core. Even more detrimental, we show that these attacks are still practicable when a rich operating system is used.

1 Introduction

The need for direct physical access to a target to perform a hardware attack was recently proved obsolete. Software-exposed hardware mechanisms implemented to improve SoC performance or power consumption were shown to be susceptible to remote hijacking by attackers seeking to perform fault injection or Side-Channel Attacks (SCAs).

Since 2014, and the *Rowhammer* vulnerability's disclosure [14], the remote attack threat has become prevalent in hardware security researches. As a matter of fact, the influx of connected devices associated with the multiplication of cloud services offers a new playing field for attackers. Moreover, despite the appearance of trusted entities (ARM TrustZone, Intel SGX) that testify a growing need for SoC security, the hardware threat remains underestimated.

Between 2014 and today, *Rowhammer* capability evolved from random bit flips generation to privilege escalation on remote devices [12,17,32]. Meanwhile, the *CLKSCREW* exploit demonstrated that power and clock glitch attacks can

be launched from within an ARM SoC using software programmable voltage-frequency regulators [28]. Recently, this attack was improved [24] and deployed on Intel SGX devices [13,21]. From a side-channel point of view, two novel families of remote attacks have been introduced. On the one hand, micro-architectural timing attacks with *Meltdown-Spectre* [19,15], *Foreshadow (SGX)* [29] and more recently *MDS* exploits [30,5]. These attacks leverage speculative and out-of-order execution in modern processors to steal secret data from victim processes. On the other hand, remote power SCAs have been introduced through several works on FPGA devices. Through the implementation of sensors inside a multi-user FPGA fabric, it was demonstrated that an adversary can eavesdrop the activity of the other users [26]. More recently, remote power SCAs have been extended to microcontroller devices using the ADCs they embed [10,22] and to Intel devices using the RAPL interface [18]. This spreads further the threats posed by remote SCAs from FPGA fabrics to general purpose microcontrollers as those found in usual connected devices.

In this paper we introduce *SideLine*, a novel side-channel vector based on the intentional misuse of hardware resources available in high-end SoC devices. *SideLine* leverages delay-lines components embedded in SoCs that use external memory; it neither requires embedded reconfigurable logic (FPGA) nor analog circuitry (ADC). Two delay-line blocks namely *delay-locked-loop* and programmable *delay-block* are hijacked to perform voltage measurements and maliciously used to conduct power SCAs on application processors (AP) and microcontrollers units (MCU). *SideLine* makes it possible for an attacker to perform software-induced hardware attacks without direct physical access to the target. Our contributions are listed below:

- We reveal that delay-line-based components available in a broad range of SoCs that employ external memories can be turned into power consumption measurement units.
- We describe three attacker-victim (core-vs-core) delay-line-based SCA scenarios over two SoC devices: **AP-vs-AP** attack (on a Xilinx Zynq 7000 SoC), **AP-vs-MCU** attack and **MCU-vs-AP** attack (on a STMicroelectronics STM32MP1 SoC) where AP and MCU respectively denote the application processor and the microcontroller.
- For each scenario a correlation power analysis attack is conducted against the publicly available OpenSSL AES encryption algorithm and the full secret key is successfully recovered. The attack feasibility is demonstrated on bare metal and Linux OS-based applications.

Responsible Disclosure: We responsibly disclosed our findings to Xilinx on September 22th, 2020 and STMicroelectronics on November 2nd, 2020. Both acknowledged and agreed on the publication of these results. Moreover, this disclosure led to a close collaboration with these companies to find and build efficient countermeasures against *SideLine* and similar attacks. Please keep in mind that *SideLine* has been performed on these two processors for demonstration purposes but the concept is generic and any devices that embed delay-lines can be affected.

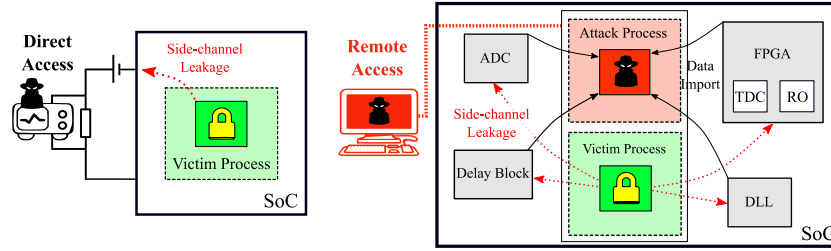


Fig. 1: On the left, local power SCA uses voltage probes to eavesdrop a leakage from a victim process. On the right, remote power SCA leverages the target’s resources to monitor the victim process leakage without requiring physical access.

Outline: The remainder of this paper is organized as follows. In section 2, we provide background information on power SCAs and describe the state-of-the-art. In section 3, we introduce delay-lines and their applications in SoC devices. Then, we present the tested products and the associated threat model in section 4. Sections 5 and 6 are dedicated to the deployment of the three attack scenarios. Finally, we discuss performance, limitations, countermeasures in section 7 and conclude in section 8.

2 Background

This section reminds the general side-channel background, the techniques recently introduced to monitor on-chip voltage fluctuations and the related works.

2.1 Power Side-Channel Attacks

A power SCA makes use of transistors switching activity leakage through power consumption variations to collect information about the processes running inside a device. Thanks to the correlation that exists between this leakage and the processed data, an attacker may try to launch an SCA to recover secret data or cryptographic keys from a target. Traditional power SCAs monitor the voltage variations induced by a device through a resistor attached to its power pads [16]. Simply by analysing the collected traces, an attacker can visually speculate on the different instructions executed by the target using a so-called Simple Power Analysis (SPA [16]) attack. Such SPA was proved effective to recover the private key used by asymmetric encryption algorithms like RSA or ECC [31]. Differential Power Analysis [16] and Correlation Power Analysis (CPA) [3] use statistical tools to infer secret keys by correlating guessed leakage hypotheses with a set of experimental traces.

Traditionally, power SCAs are carried out locally, in laboratories, using a voltage probe and an oscilloscope as depicted by the direct physical access attack path in Figure 1. These attacks target secure integrated circuits, such as smart cards or cryptographic accelerators embedded in SoCs. SCA countermeasures such as masking, jitter or shuffling [34,36] are usually implemented in such

secure devices. It encourages the use of high resolution and high sampling rate oscilloscopes on the attacker side to outperform the countermeasures.

Because traditional hardware attacks are assumed local and expensive, a large number of electronic devices are not prepared to withstand remote hardware attack scenarios. For this reason, even with limited performances, digital and analog integrated sensors may manage to jeopardize the security of devices ranging from IoT components to cloud servers (remote access in Figure 1). With the advent of these software-induced hardware attacks that do not require either direct physical access to the target or specific equipment, the alleged hardware attack limitations are called into question or even removed.

2.2 On-Chip Voltage Sensing

Two families of sensors enable malicious on-chip voltage sensing: either delay sensors built with digital logic gates which aim at measuring fluctuations in the power consumption through delay variations [37,38], or analog sensors using ADCs usually embedded in MCUs [10,22]. Until this work, digital sensors dedicated to SCAs have been exclusively implemented in FPGAs. Their available programmable logic makes it possible to design and tune such delay sensors in order to measure the power consumption of a device. We describe hereafter the principles of these delay sensors as their working principle is similar to the delay-line components we used.

Delay-based voltage sensors leverage a side-effect of voltage fluctuations over digital logic behavior, which is the relationship between the time taken by a signal to propagate through a digital logic gate and the on-chip voltage level. An increase of the gate’s power supply translates into a shortening of its propagation delay, and respectively a reduction of the voltage induces its increase [9]. As a result, measuring the variations of the logic gates propagation delay provides an image of their voltage supply variations. Temperature and capacitive effects also play a significant part in its equation [9]. Unlike voltage, the propagation delay can be directly measured using digital logic. Commonly used FPGA-based sensors are the Ring-Oscillator (RO [37]) and the Time-to-Digital Converters (TDC [26]).

2.3 Related Works

In 2018, Schellenberg et al. demonstrated that FPGA-based sensors were precise enough to be used for SCAs on public and secret cryptographic algorithms [26]. To enable this attack, the adversary (a TDC-based delay sensor and its control logic for power supply measurement) and the victim (an AES hardware encryption block) needed to be located within the same FPGA. We define it as an **FPGA-to-FPGA** attack. The associated threat model targets multi-user FPGA cloud services that may appear over the next few years [6]. The same year, Zhao et al. disclosed that power SCAs can be conducted on heterogeneous platforms that include both an application processor and an FPGA fabric on the same silicon die. As a proof of concept, they were able to successfully retrieve

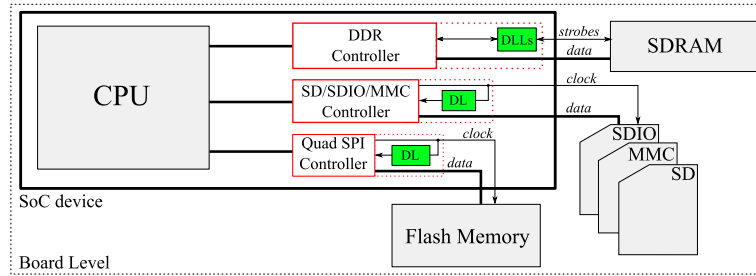


Fig. 2: Typical SoC connectivity with external memories. Delay-lines are implemented to synchronize clock and data signals arrival in the memory controllers.

the secret key of a custom RSA implementation running within a CPU core [35]. To do so, they carried out an SPA attack using RO-based voltage sensors implemented in the FPGA fabric.

Until 2019, remote power SCA remained bounded to FPGA devices or heterogeneous SoCs embedding an FPGA fabric as its flexibility allowed the implementation of powerful sensors. Two works went beyond the FPGA by proving that on-chip power SCAs can be carried out in microcontroller devices [10,22]. These attacks use ADCs as a straightforward way to measure on-chip power supply level. Thanks to a leakage of the chip power consumption into this analog block, the ADC can substitute the voltage probe role. Even with an extremely limited sampling rate, this noise sampling method was successful in retrieving the secret keys used by real world software and hardware AES cryptographic libraries.

3 Delay-Lines in High-End SoC Devices

Delay-line-based sensors were previously used in FPGA devices as a way to monitor chip power consumption (TDC sensor). Despite offering great performance, these sensors were limited to configurable logic which is rarely integrated in SoC devices. In this section, we disclose that digital and analog delay-lines are widely implemented in SoC memory controllers. We present them and discuss their potential use as voltage sensors (delay sensors).

3.1 Memory Controller Basics

Because high-end SoCs are designed to run operating systems (Linux, Android, etc.), they require a large amount of Non-Volatile Memory (NVM) to store the OS and Random-Access-Memory (RAM) to efficiently load it. Due to technological constraints, these SoCs do not embed a significant amount of RAM nor NVM memory but are rather interconnected with external memories (memory cards, Flash memory, SDRAM memory, etc). Thus, depending on the form-factor, speed and memory size constraints, designers can choose between a wide

range of external memory devices. A typical scenario of a SoC using external memories is depicted in Figure 2.

Several memory controllers are required to interface the SoC with its external memories. Each memory controller acts as a request arbiter, a transaction scheduler and as a physical interface to manage data flowing from the SoC to the memory, and vice-versa. In embedded systems, for cost and efficiency reasons, the memory controller is more likely to be directly integrated as a part of the SoC. At the edge of the memory controller, a physical controller (dotted lines in Figure 2) outputs and captures the signals that will flow between the SoC I/Os and the memory device I/Os (clock, data, configuration signals, etc.). The physical controller also ensures that these signals arrive on time regardless of the interconnection tracks length on the PCB, the voltage and the temperature variations. To better understand the extent of memory signal propagation timings, we draw a simple example of SoC/Synchronous Dynamic-RAM (SDRAM) association. When a read operation is initiated by the SoC, the external SDRAM memory outputs the requested data edge-aligned with a clock signal (strobe) later dedicated to data sampling. Depending on the PCB tracks length, the clock signal is likely to shift ahead of the data signals, leading then to a sampling error. To mitigate this effect, the SoC physical controller implements delay-line-based components (delay-locked-loop *DLL* and programmable delay-block *DL* in Figure 2) to calibrate the phase alignment between the sampling clock and the data signals. This calibration can be manual and made once and for all after testing at manufacturing or performed at each chip power-up. It can also be adjusted dynamically to counterbalance any misalignment due to power supply or temperature fluctuations.

The relationship between the delay applied and the SoC voltage fluctuations drew our interest. In the following paragraphs, we present two different delay-line-based mechanisms that can be used to generate these delays for low and high-bandwidth external memory applications.

3.2 Delay-blocks in Low-Bandwidth Memory Controllers

In relatively low-bandwidth external memories such as Flash memories, SD cards and multimedia cards, the impact of voltage and temperature fluctuations is considered not significant enough to jeopardize the communication integrity: dynamic calibration is not required. Delay-lines are nonetheless used to mitigate the impact of the PCB track length on the data and clock signals propagation timings (these delays are not predictable by SoC designers, they are set only at board design time). As track lengths are fixed, a static delay is sufficient to ensure good operation. For a read transaction, the delay-line is typically calibrated in order to add a phase shift of 90° to the clock signal. Thus, it ensures that data signals are in place when sampling occurs. The delay-line calibration is carried out through a series of training steps. These training steps modify the delay of the elements forming the chain and, for each configuration, verify if the external memory has been properly read. If the training is successful, the

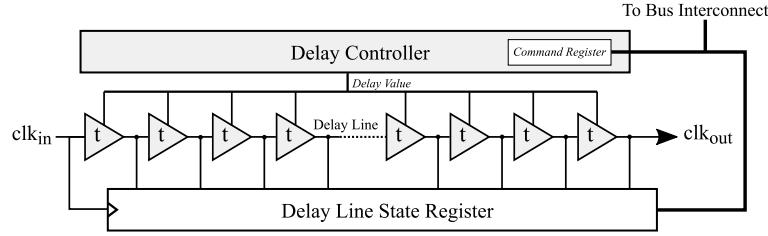


Fig. 3: An example of delay-block used in low-bandwidth memory controllers.

delay-line configuration is saved in a dedicated register and remains unchanged until the next test.

Several SoC vendors provide user programmable delay-blocks as a way for developers to be able to use a wide range of memory chips or cards with different bus speeds. Unlike traditional static delay-lines, these delay-blocks come with both a complete calibration toolkit and a detailed documentation. Figure 3 illustrates the delay-block structure that was observed in one of the SoC we benchmarked. Its purpose is to delay the clock signal with respect to the data signals when a read operation is conducted. The block consists in a simple delay-line associated with a set of control and status registers. A *Command Register* controls the delay t of all the delay-line elements and thus the phase shift added to the clk signal. To ensure that the phase shift obtained is conform to the applied command, a *state register* captures the output of each element forming the delay-line every time a clk_{in} rising edge event occurs. Then, a specific training is performed to verify whether the captured pattern matches the command or not.

Despite some missing parts, this structure is reminiscent of that of a TDC as the delay-line state is continuously captured and stored in an accessible register. In section 6, we demonstrate that this delay-block can be turned into a voltage sensor and hijacked to perform a power SCA.

3.3 DLLs in High-Bandwidth Memory Controllers

Because of the continuous increasing in memory bus speeds, the available slack time for data sampling is gradually shrinking. Double data rate memories (DDR) such as SDRAM memory perform one data transfer per clock edge (both rising and falling) while reaching gigahertz frequencies [25]. On these devices, the data sampling is very likely to get corrupted by temperature and voltage variations. This time, a static delay source is not suitable to ensure correct operations. To effectively cancel voltage and temperature noise side-effects, a dynamic way to adapt the clock delay has to be considered.

Delay Locked Loops (DLLs) are generally used in recent DDR memory controllers to dynamically track and control the phase shift applied between the sampling clock and the external memory (e.g. SDRAM) data signals [2,7]. As illustrated in Figure 4, a DLL has two main blocks: a delay-line, and a feedback

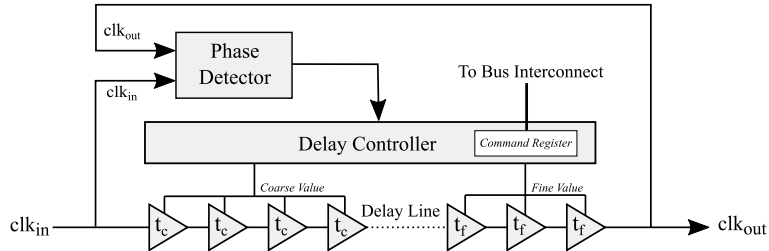


Fig. 4: An example of delay-locked-loop used in DDR memory controllers.

circuit. The delay-line is calibrated to provide a phase shift to a clk signal using both *coarse* and *fine* delay elements. However, the propagation delay jitter associated with on-chip voltage and temperature fluctuations is likely to skew the applied phase. This is why a DLL includes a feedback circuit to tune the delay-line in order to provide a dynamic control of the phase shift and thus, counterbalance voltage and temperature variations. The feedback circuit comes with a phase detector that compares the phase shift between the clock signal at the input of the delay-line, clk_{in} , and its phase-shifted clock output, clk_{out} . Then, according to the measured error, a delay controller applies a correction in order to "deskew" the result, that is, to get back to the initial delay. The applied correction modifies the delay of the elements forming the delay-line and can be either analog or digital-controlled depending on the delay-line type [1].

A *command register* stores the delay settings, it is memory-mapped and hence can be read from the SoC AP or MCU cores. The DLL operates autonomously, this means that through a simple access to this register, a process can retrieve the state of the DLL, which shall be correlated to on-chip voltage and temperature variations. As a result, tracking the *command register* content shall provide an image of the SoC power consumption that may be used to carry out SCAs. Note that this measurement methodology (tracking the command of a feedback dynamically controlled system) differs from that described in Section 3.2 for delay-blocks (sampling a clock signal propagating inside a fixed delay-line). If this unusual measurement medium provides enough resolution and sampling rate to eavesdrop power consumption of secure applications running on a processor, this could represent an important backdoor for computer security. This hypothetical vulnerability is strengthened by the fact that this attack only requires a read access to the command register, no configuration steps are required. This attack scenario is developed in section 5.

4 Experimental Setup

4.1 Tested Devices

Two devices from two different SoC providers have been studied in our experiments. The first target considered in this work is a Xilinx Zynq-7000 SoC [33] that comes with a dual-core Cortex-A9 application processor (AP). It is a typical

multi-purpose SoC providing many additional resources: FPGA, I/O, ADCs, bus controllers, etc. It supports DDR2-DDR3, Flash and SD/MMC external memories and provides several DLL blocks to interface properly with DDR external memories. The experiments made on this target have been conducted without using an OS: we denote it as a **bare metal attack**. This configuration makes SCA easier as there are fewer interruptions (with respect to the case in which an OS is used) that may disturb the attack and victim processes and cause synchronization issues. The entire Zynq-based *SideLine* attack code can be cloned from GitHub: https://github.com/Remote-HWA/SideLine_Zynq.

The second target is a STMicroelectronics STM32MP157C-DK2 development board [20] that embeds a dual-core Cortex-A7 AP associated with a Cortex-M processor (MCU). It also supports DDR2-DDR3, Flash and SD/MMC external memories and embeds several DLL blocks. Additionally, it provides user programmable delay-blocks (DLYB [20]) that can be employed for interfacing low bandwidth memory (e.g. an SD card). These programmable delay-blocks are the second case we studied. The experiments done on this SoC have been carried out with a Linux OS running on its AP (i.e. the Cortex-A7 processor). The results are those of a **Linux OS attack**. The entire STM32MP1-based *SideLine* attack code can be cloned from GitHub: https://github.com/Remote-HWA/SideLine_STM32MP1.

4.2 OpenSSL AES Architecture

The OpenSSL library [23] provides several cryptographic algorithms used for securing channels over computer networks. In this work, we focus on the OpenSSL AES-128 (version 1.1.1) that implements a 32-bit tabulated version of the textbook AES encryption algorithm [8]. This variant merges the *Mixcolumn* and *SubBytes* transformations into 4 pre-computed look-up tables known as T-tables (256 x 32-bit) as a way to optimize the computations on 32-bit processors.

4.3 Threat Model

In this work, we introduce three core-vs-core attack scenarios in order to assess the SCA capabilities of the delay-line-based sensors. For each scenario depicted in Figure 5, we first deploy a cryptographic application (in green) within a processor core. This application located either in the AP or in the MCU allows the end-user to launch AES encryptions/decryptions, with the plaintexts/ciphertexts that he provides. Secondly, we introduce a malicious user (in red) that has the privilege level necessary to access the delay-line blocks presented in Section 3 and that uses them to retrieve the leakage induced by the AES application.

Although not used in this research work, Trusted Execution Environment (TEE) and TrustZone [2] architecture stand as potential realistic targets for the delay-lines. TrustZone attacks from the normal-world to the secure-world have been widely covered in recent remote attack works [28,24,4,22]. However, from a side-channel point of view, the current TrustZone does not provide any

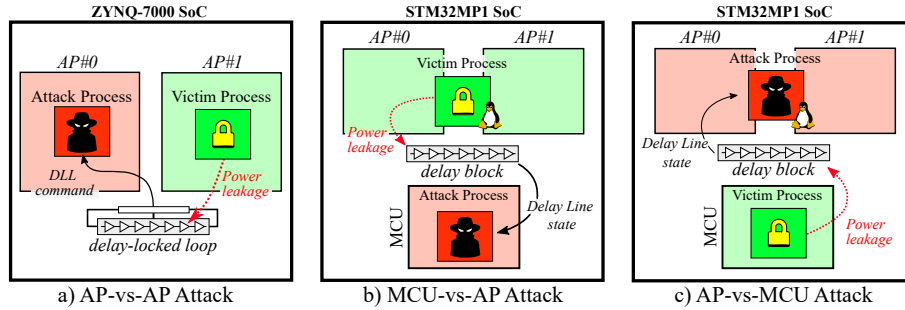


Fig. 5: Basic principle of the three core-vs-core attack variants proposed in this work. It shows the leakage path from the victim process to the delay sensor and the sensor data flow retrieved by the attack process.

countermeasures. Thus, the ability of an attacker to turn our feasibility attack into an end-to-end TrustZone attack is reasonably expected.

In the remainder of the paper, the three scenarios presented are referred to as:

1. A **DLL-based attack** (Figure 5.a), or AP-vs-AP attack, that demonstrates the ability of a DLL to serve as a power supply sensor suitable for a CPA attack against the AES algorithm. In this scenario, one core of the Zynq processor runs the AES victim application, while the second core executes the attack process (both victim and aggressor processes are C programs, in bare metal mode). The attacker code is in charge of collecting the leakage data of the AES. It does so by configuring the access to the DLL command register that makes it possible to sample its values during AES encryptions performed by the first core. The attacker core is also in charge of providing the plaintext to be ciphered by the victim process and to trigger both the encryption and readback of DLL states. This AP-vs-AP attack scenario is described in details in Section 5.
2. A first **Delay-Block-based attack** (Figure 5.b), or MCU-vs-AP attack, where the victim process is ran on the STM32MP1 AP (a C code AES running on top of a Linux OS) and the attack process is executed by the Cortex-M MCU (a C program, in bare metal mode). In this scenario the MCU is in charge of calibrating and using a delay-block to eavesdrop the activity of the AP. This MCU-vs-AP attack scenario is addressed in Section 6.
3. A second **Delay-Block-based attack** (Figure 5.c), or AP-vs-MCU attack, that matches a typical state-of-the-art industrial case where the cryptographic and security operations of a SoC embedding AP cores are delegated to a less complex MCU core. In this scenario the AP core (Cortex-A7) runs the attack process while the MCU core (Cortex-M) runs the AES victim process. This AP-vs-MCU attack scenario is reported in Section 6.

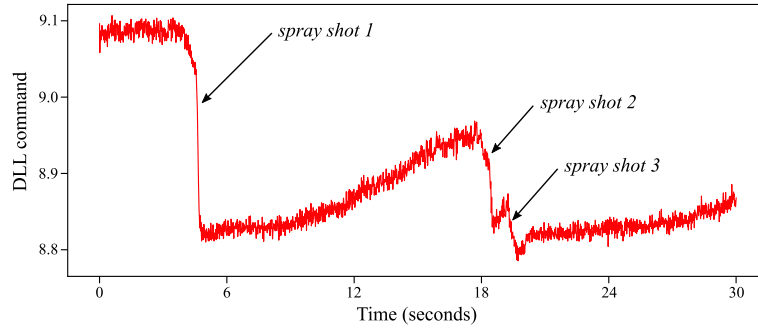


Fig. 6: DLL response to sudden temperature drops induced by three successive exposition of the SoC to a cooling spray.

5 DLL-based Power Side-Channel Attack

This section presents a novel way to monitor on-chip voltage fluctuations and conduct power SCAs using the DLLs embedded in SoC memory controllers.

5.1 Validating DLL Effectiveness: *Monitoring Temperature*

As a proof of concept, a simple experiment was carried-out on the Zynq SoC to confirm that the DLL command is actually tracking the SoC package temperature variations. The test uses a C program designed to continuously read and store the DLL command register content into an acquisition array for a period of 30 seconds. Simultaneously, a cooling spray was used at specific moments to cool down the SoC package. To limit the acquisition size, each array index contains the average of 1,000 successive DLL readings. Figure 6 reports the evolution of the measured DLL command (y-axis) as a function of time (x-axis). Each spray shot induces a temperature drop (translated into a DLL command drop in Figure 6) that progressively recovers until the next one. This simple experiment confirms that a DLL is suitable to dynamically track the SoC temperature variations. As the temperature decreases, the propagation speed of the clk signal through the delay-line increases [9]. Thus, the phase-shift between clk_{in} and clk_{out} progressively drifts. To counterbalance this effect, the DLL dynamically adapts its command in order to maintain a constant phase shift. Because package temperature evolves relatively slowly, the sampling frequency for this experiment was limited to 300 kHz. However, as this paper focuses on power side-channel, which itself depends on transient voltage drops measurements, a higher sampling rate needs to be achieved: it is the subject of the next subsection 5.2.

5.2 Improving Sampling Rate and Synchronisation using DMA

As mentioned before, the DLL command value can be directly accessed through its memory address. Then, a loop associated with an array can be added to collect more samples. This CPU-based sampling method works in principle but has several drawbacks:

First, it requires a constant time between each acquisition. If this constant time is not achieved, the samples won't be correctly aligned. Consequently, statistical attacks will be less accurate as the averaging of several acquisitions will suffer from de-synchronisation. Achieving constant time is feasible in bare metal applications because they rarely suffer from interruptions. However, if the application runs over an OS, interrupts will dramatically affect the timing of acquisitions and make their averaging impossible. The second limitation is related to the achievable sampling rate. Indeed, the delay induced by CPU memory access plus the storage of the acquired data into an array is not optimal. Using this method on the Zynq SoC, the sampling frequency was limited to 2.2 MHz.

To solve these issues, we choose to use Direct Memory Access (DMA) in order to improve the sampling rate as well as the synchronisation of our samples (as proposed in [10]). A DMA is a hardware module able to transfer data from a peripheral to another without processor intervention. For this reason, it is faster in transmitting data, but also not affected by OS interrupts. The source address (address from which the DMA should sample the data) is the register containing the DLL command. The destination address (destination of the DMA transfer) is the base address of an array whose size depends on the number of samples required. At the end of the DMA transfer, an interrupt flag is set and ends the sampling process. With DMA up and running, we improved the DLL sampling frequency from 2.2 MHz to 16 MHz.

5.3 Bare Metal OpenSSL AES Attack Setup

According to the threat model we consider (see subsection 4.3), the attack process shall be able (1) to trigger the start of an AES encryption by the victim process, and (2) to control the gathering of the leakage from the AES through a DLL-based voltage sensor. Our test bench includes two processes (their pseudo codes are given in appendix 1 and 2) executed by the two application cores of our target in bare metal mode: the attack process on AP#0 and the victim process on AP#1.

In addition to this attack setup, we used embedded hardware performance counters to precisely measure the duration of an AES encryption. On average, an encryption took 837 AP clock cycles or $1,25 \mu\text{s}$ at a frequency of 667 MHz (both attack and victim programs were compiled with the optimization parameter set to -O2). The DMA transfer method we used provides a constant 62.5 ns sampling period (i.e. a 16 MHz sampling frequency). As a result, 21 samples of the DLL command are gathered per AES encryption.

5.4 DLL-based SCA Attack on Zynq SoC

The bottom part of Figure 7 illustrates the results of two experiments conducted to assess the AES encryption impact on the DLL command value and precisely detect its encryption time window. The two traces depicted in black (1st case) and red (2nd case) represent the averaged DLL command value (y-axis) obtained for 1,000 acquisitions as a function of time (expressed in DMA samples). For the

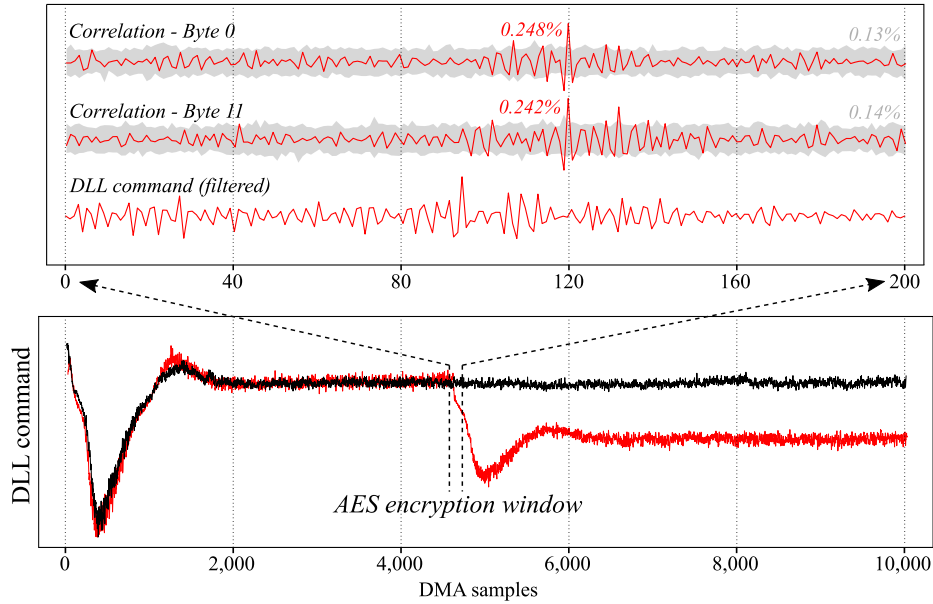


Fig. 7: DLL-based attack results: the bottom part represents the impact of an AES encryption on the DLL command value. The top part zooms on the AES encryption windows and provides the temporal correlation rate for two key bytes.

first experiment (in black), the victim program was kept idle during the entirety of the DMA sampling operations. The DLL command drop visible between sample 0 and 1,000 was induced by the extra power consumption linked to the DMA module activation. The DLL applied a strong correction to maintain a constant phase shift, that was finally relaxed as the power consumption returned to normal (sample 2,000 to the end of sampling). The second case (in red) reports an actual iteration of the attack and victim processes when an AES encryption is done. The red trace experienced the same DLL command undershoot due to DMA module activation (sample 0 to 1,000) but also a second undershoot corresponding to the AES encryption (starting at sample 4,500). It is finally restored to a steady value lower than the initial one (sample 6,000 to the end of sampling). The AES encryption window was deduced from the position of the second DLL command drop. Based on this information the CPA attack could be conducted on a smaller amount of samples.

We launched a total number of 20 million AES encryptions and acquired 200 DLL command samples per encryption. Samples and plaintexts extraction through UART took around 8 hours at 921,600 bauds. Then, an external computer was used to apply post-processing to the traces and conduct the CPA attack. The top part of Figure 7 depicts a filtered and averaged trace of the DLL command (in red). High-pass filtering was used as a way to reduce the impact of low frequency variations (induced for instance by temperature fluctuations) on the acquired traces and thus to reduce the number of traces required for the at-

tack. Then, we performed a plaintext-based CPA attack on the first round of the AES. As we mentioned earlier the OpenSSL AES uses T-tables to upgrade its performances on 32-bit processors. This allows us to leverage a 32-bit T-tables output prediction: $HW[T_{table}(key \oplus plaintext)]$. The obtained correlation results versus the time are represented above the averaged trace in Figure 7 (for two key bytes). The correct key hypotheses are depicted in red and emerge from the incorrect hypotheses (in grey) at sample 120. Based on 20 million encryptions, we achieved a full AES key recovery. 3 bytes were retrieved in the range 0-5M traces, 2 between 5-10M million, 5 between 10-15M and 4 between 15-20M. The key bytes number 7 and 9 never completely emerged from the incorrect candidates, but we assume that a simple brute force can be conducted to retrieve their values. The progressive correlation of the first 8 key bytes plus the failed byte #9 are depicted in Figure 12 in the appendix.

5.5 Conclusion on DLL-based SCA

In this section, we demonstrated that a DLL can be used to monitor on-chip temperature and power supply fluctuations. This unconventional voltage sensor was then used to conduct a power SCA on an OpenSSL AES algorithm implemented in the Zynq application processor and a full AES key recovery was achieved (with the help of brute force for the two remaining bytes). Performance, limitations and potential countermeasures regarding this attack are discussed in Section 7.

6 Delay-Block-based Power Side-Channel Attack

The DLL-based attack presented in Section 5 was associated with the use of DDR external memories such as SDRAM in AP-based SoC. This section discloses a second attack path that allows the hijacking of a programmable delay-block and its malicious use to perform core-vs-core power SCAs. These experiments are conducted on the STM32MP1 SoC.

6.1 From Delay-Block to TDC Sensor

The STM32MP1 SoC comes with three programmable delay-blocks IPs (DLYB [20]) capable of working with different types of external memories (QSPI, SD, MMC). Their settings can be adjusted depending on the bus speeds of the external memories used. Their initial purpose is to adjust the phase of the clock signal in order to ensure a reliable exchange of data by tuning the clock delay.

The left part of figure 8 depicts the 12 elements delay-line provided by the STM32MP1 delay-block and the capture register designed to monitor the state of the output nodes of every delay element. When a clk_{in} rising edge occurs, the capture register takes a snapshot of the delay-line. This snapshot contains an image (represented as a waveform in Figure 8) of the clock propagation through the delay-line. The propagation delay t of the elementary delay elements can

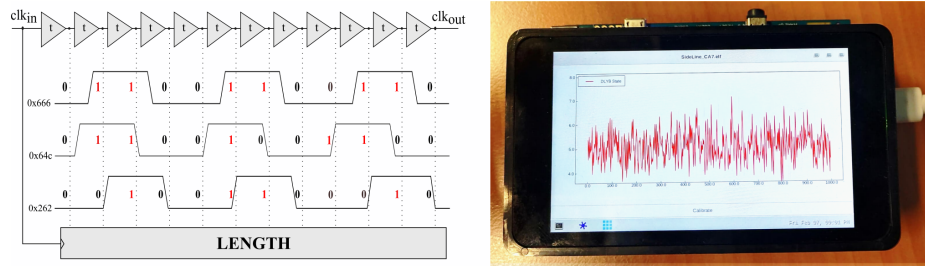


Fig. 8: Effect of on-chip voltage variations on the sampled delay values.

be set using a dedicated register. If this delay is set to its minimum the delay-line width (acquisition window) is small. Thus, only a part of the clock signal can be captured. By gradually increasing t , the clock signal observation can be extended, possibly to several periods.

We leveraged this t parameter to make the delay-block sensitive to on-chip voltage fluctuations. To that end, we took a significant number of delay-line snapshots for each of the 128 possible t delay values. A vast majority of them gave stable results; which means that the captured image remained stable between successive register readings. For a few however, delay variations arose between subsequent captures. This interesting behavior can be explained by (1) on-chip voltage fluctuations that affect the clock propagation time through the delay elements, and (2) by the fact that several delay values t naturally position the clock edges in unstable places within the delay line (i.e. in between two delay elements). The left part of figure 8 displays three waveforms (delay-line snapshots) obtained with such a t setting. In this configuration, three clock periods stand in the entire delay line. From top to bottom we have: (1) the steady state register waveform which stands as our reference (it outputs a 0x666 reference value), (2) a slowed down waveform that can be obtained due to a supply voltage decrease (it outputs a 0x64c), and (3) an accelerated waveform that can be obtained due to a supply voltage increase (it outputs a 0x262). In our experiments, the three obtained hexadecimal digits are weighted and added to translate into an image of the voltage supply.

On the right part of figure 8, a program displays as an oscilloscope the actual delay-line state on the STM32MP1 touchscreen. This way, the actual power consumption noise impact on the delay-block state can be directly observed. To make it possible, the implemented program automatically calibrates the delay-block by testing various delay parameters. For each delay value, it collects multiple delay-line state samples, computes their variance and adopts the calibration that provided the highest variance. Indeed, a higher variance indicates an important delay instability and thus a stronger relationship with voltage fluctuations.

6.2 Linux-based OpenSSL AES Attack Setup

Similarly to the attack setup described in subsection 5.3, we used the OpenSSL AES implementation to evaluate the threat posed by delay-block-based SCAs.

The STM32MP1 embeds both a dual core AP and a MCU that makes it possible to test the MCU-vs-AP and AP-vs-MCU attack scenarios introduced in subsection 4.3. Depending on the scenario, the attack and victim processes were ran either on the AP core or on the MCU core. Here, we consider the MCU-vs-AP attack to describe our attack setup.

We use an adapted version of the Zynq-based attack. On the adversary’s side (here the MCU), delay-block calibration and use of Hardware Performance Counters (HPCs) were added to the initial algorithm. HPCs are used to accurately time the successive encryptions and to mitigate the de-synchronisation brought by the Linux OS. For each acquisition, the number of cycles elapsed during the encryption is compared to a maximal limit Nb_{cycle} set by the adversary above which the entire acquisition is discarded. Prior to the attack, a preliminary test was conducted in order to identify the optimal value for Nb_{cycle} (assuming that a lower number of clock cycles corresponds to a lower number of interrupts). Hence, by launching thousands of AES encryptions, we were able to find a reference number of clock cycles for almost interrupt-free encryptions. Then, based on this reference, we set a maximal limit Nb_{cycle} beyond which we decided to discard the acquisitions. By doing so, at least half of the total acquisitions were retained and used for the subsequent CPA calculations.

Regarding the CPA, we embedded it directly within the STM32MP1. This way, we drastically limited the amount of data exported. Moreover, this allowed us to directly plot the results on screen as illustrated in appendix figure 11.

6.3 Delay-block-based SCA Attacks on STM32MP1 SoC

In the AP-vs-MCU attack scenario, the OpenSSL AES program runs within the STM32MP1 Cortex-M MCU. Using compiler optimization set to -O0, 1,460 clock cycles are required to perform a single AES encryption, that is $7.3 \mu\text{s}$ at the MCU operating frequency (200 MHz). Figure 9 displays in its bottom part the averaged delay values obtained for a time window of 250 DMA samples (or $16.4 \mu\text{s}$) over 10 million acquisitions. The AES encryption, which approximately covers 110 DMA samples, is surrounded by two empty `for` loops added for visualisation ease. The top part of Figure 9 provides the CPA correlation rates of four key bytes (of index #1, #13, #9, and #5) as a function of time. The correct key hypotheses are depicted in red and emerge from the incorrect hypotheses (in grey) between samples 70 and 80. We chose to represent these key bytes because they are equally distant regarding the OpenSSL byte computation order: *0 5 10 15 - 4 9 14 3 - 8 13 2 7 - 12 1 6 11*. This explains the regular temporal offset observed between them. Based on 10 million encryptions, we achieved a full AES key recovery. 6 bytes were retrieved in the range 0-2M traces, 4 between 2-6M and 6 between 6-10M. The progressive correlation of the eight last AES key bytes (#8 to #15) are depicted in Figure 13 in the appendix.

In the MCU-vs-AP attack scenario, the OpenSSL AES program runs in the STM32MP1 Cortex-A7 AP. Using compiler optimization set to -O2, 865 clock cycles are required to perform a single AES encryption, that is $1.33 \mu\text{s}$ at

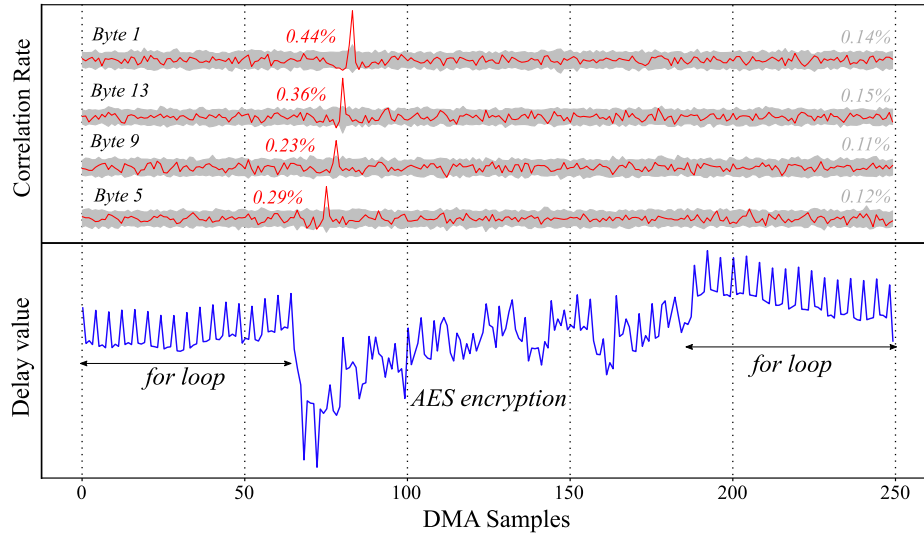


Fig. 9: AP-vs-MCU attack results: the bottom part represents the averaged AES power consumption, the top part provides the correlation rates as a function of time for four AES key bytes.

the AP operating frequency (650 MHz). Figure 10 displays in its bottom part the averaged delay value obtained for a time window of 100 DMA samples (or $6,6 \mu\text{s}$) over 40 million acquisitions. The AES encryption, which approximately covers 20 DMA samples, is surrounded by two empty `for` loops added for visualisation ease. The top part of Figure 10 provides the temporal correlation rate of four key bytes as a function of time. The correct key hypotheses are depicted in red and emerge from the incorrect hypotheses (in grey) between samples 30 and 40. Again, we chose to represent these specific key bytes because they are equally distant in the OpenSSL byte computation order. However, the AES encryption in the AP is faster than that of the MCU ($1.33 \mu\text{s}$ vs. $7.3 \mu\text{s}$) and the DMA sampling frequency that remained fixed between the two experiments is no longer sufficient to let the temporal offsets appear. This limited sampling frequency partly explains the higher number of acquisitions required to retrieve some key bytes. For instance, byte #12 in Figure 10, seems to suffer from the under sampling and gave poorer correlation results (0,07%) than byte #4 (0,32%) or byte #0 (0,29%). We were able to confirm this assumption through a second experiment where the AES encryption temporal window had been slightly shifted regarding the DMA: the AES leakage was thus sampled at different timings. This experiment gave better results on several key bytes that struggled to emerge in the previous attack. Based on 40 million encryptions, we achieved a full AES key recovery. 3 bytes were retrieved in the range 0-10M traces, 6 between 10-20M, 2 between 40-30M, 4 between 30-40M. The 13th key byte never completely emerged from the incorrect candidates, but we assume that a simple brute force can be conducted to retrieve its value. The progressive correlation of the first key bytes (0 to 7) are depicted in Figure 14 in the appendix.

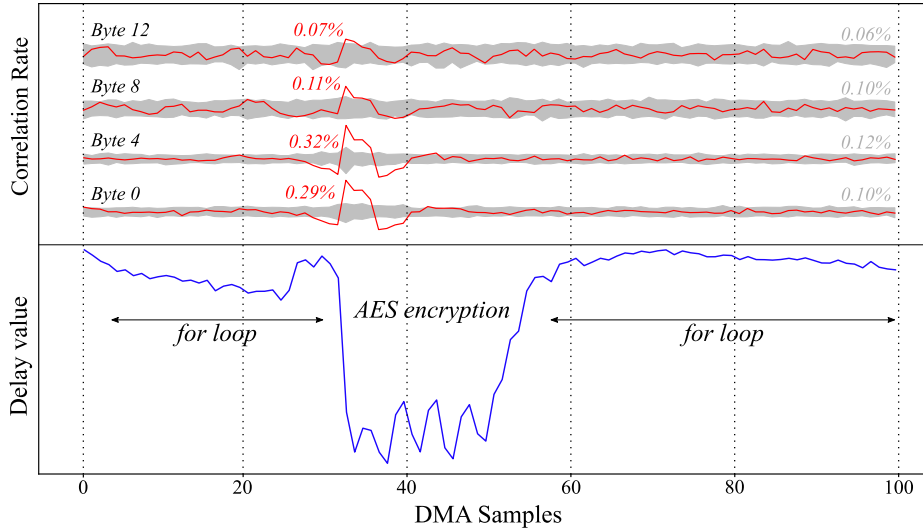


Fig. 10: MCU-vs-AP attack results: the bottom part represents the averaged AES power consumption. The top part provides the correlation over the time results over four AES key bytes.

7 Discussion

Two delay-line-based power measurement techniques, using a DLL or a delay-block were introduced and studied in this research work. Because such delay-line-based components are embedded in almost every high-end digital SoC that uses external memories, the threat model we introduced is serious and shall be considered feasible for a large number of complex SoCs. In this section, we discuss performance, additional attack scenarios and potential countermeasures regarding the *SideLine* attack.

7.1 Performance and Limitations of *SideLine*

Table 1 summarizes the results obtained for the three attack scenarios considered in this paper. First, an AP-vs-AP attack was performed on a Zynq SoC using DLL-based sensors. As DLLs provide a limited resolution, a large amount of acquisitions were required to integrate enough information for the CPA to succeed (20 million traces required for full AES key recovery). It took around 12 hours to extract the traces, apply post-processing (filtering) and conduct the CPA attack. The lack of resolution also made post-synchronization nearly impossible and thus implied the collection of leakage traces with a constant synchronization. Apart from performances, the DLL was by far the simplest sensor to implement in our experiments, as it only required the reading of a memory-mapped register. However, care must be taken as in certain cases, DLLs may require additional calibration. For instance, some DLLs can either perform delay

Scenario	Sensor	Nb_{Acq}	$freq_{DMA}$	$freq_{Target}$	Duration
Zynq AP-vs-AP	DLL	20M	16 MHz	667 MHz	~ 12 h
STM32 AP-vs-MCU	DL	10M	15.2 MHz	200 MHz	~ 9 h
STM32 MCU-vs-AP	DL	40M	15.2 MHz	650 MHz	~ 24 h

Table 1: Overall delay-line-based power SCA results.

calibration continuously or at a set of intervals [2]. Such parameters should be taken into account by the attacker and calibrated if needed.

The second attack proposed in this paper required a preliminary work to properly turn the delay-block into a custom TDC. Then, two delay-block-based power SCAs were conducted on a STM32MP1 SoC. The AP-vs-MCU AES attack took around 10 million traces for a full key recovery (trace acquisition and CPA took approximately 9 hours) while the MCU-vs-AP AES attack required 40 million traces (24 hours). We can compare these results to the attack reported in [11] against an OpenSSL AES implementation in an FPGA-based heterogeneous SoC. In this work, FPGA-based TDCs were able to perform a similar attack using only 90,000 traces (FPGA-to-CPU attack). FPGAs indeed offer the possibility to design high resolution and high sampling rate sensors which explain the higher efficiency of their attack. Such a flexibility is obviously not available in ASICs. For instance, even using DMA in our experiments, the maximum sampling rate achieved (16 MHz) was still way under the FPGA-based TDC sampling rate given in [11] (200 MHz). Additionally delay-blocks also suffer from a poor resolution as evidenced in Figure 15 in the appendix. Despite these limitations, we demonstrated that such an attack is still feasible without using FPGAs and within a reasonable time and number of traces.

The presence of DLLs and programmable delay-blocks is already mandatory in high-end SoC devices and should become even more prevalent in the future with the constant increase of memory bus speeds. At the same time, their voltage sensing capability will be progressively enhanced as they will need to meet higher performances requirements. This should make *SideLine* even easier to conduct and detrimental for hardware security in the future.

7.2 Hardware & Software Mitigations

This section provides some countermeasure guidelines mitigating *SideLine*:

Adding SCA Countermeasures: A simple way to make the victim process more resilient to power SCAs is the addition of software or hardware SCA countermeasures [36,34]. As mentioned above, one of the main limitations of *SideLine* comes from the low resolution provided by DLL and delay-blocks. This forces the attacker to acquire a huge number of traces (several million in our case) and makes it nearly impossible to re-synchronize SCA traces. On the victim side, software randomization could be a good candidate to efficiently de-synchronize computations and hence to increase significantly the attack difficulty (e.g. adding random delays in T-Table computations for OpenSSL AES). On the monitoring side (delay-line), a straightforward way to mitigate the attack could rely on the

addition of phase and frequency jitter to the clock signal used for sampling the delay-line registers.

Preventing Delay-Line Access: Another countermeasure would act at system level by preventing the access to the delay-line registers by unauthorized software entities. Hence, only the OS for instance would have access to this resource. TrustZone could also be used to place DLLs and Delay-blocks in the secure world and make their use by non-secure world impossible in practice. Locking the access to the DMA module or the hardware performance counters would also represent a significant limitation for the attack setup.

Reducing Delay-Line Sampling Rate: Preventing delay-line access through privilege rights seems insufficient as a malicious attacker or a compromised OS could overpass it (privileges escalation). A hardware way to mitigate the threat would be to limit the delay-block access to a lower sampling rate (e.g. 10KHz). This could be simply achieved by limiting the access rate to the register that stores delay-line information. This way, even if the power consumption monitoring would remain feasible, it will highly affect the delay sensor performances. With such a limited sampling rate it would be probably very challenging for an attacker to conduct SCAs on fast encryption algorithms such as AES.

Abandoning Delay-Lines in SoCs: As *SideLine* revealed their potential misuse as power consumption sensors, the delay-line-based components could be removed from SoC devices and instead, be placed directly within the external memory devices. This drastic choice would require the addition of configuration I/Os in external memories to efficiently calibrate the delay-lines but will almost entirely remove the delay-line threat from the SoC die. However, even outside the SoC, the delay-line threat may remain problematic as inter-chip power SCAs have already been shown feasible [27].

8 Conclusion

Previous works demonstrated that remote power SCAs were feasible using FPGA-based delay sensors and microcontroller ADC-based sensors. *SideLine* goes further by proving that unsuspected hardware components available in a broad range of high-end SoC devices, can be turned into power consumption measurement units. In this work, we studied two common SoC resources known as delay-locked-loops and delay-blocks and proved their capability to eavesdrop the voltage activity of cryptographic programs running in different processors. Several core-vs-core attack scenarios on application processors and microcontroller units were conducted. For each scenario, we achieved a full key recovery side-channel attack on the publicly available OpenSSL AES implementation. We believe that these findings open a new era for remote power side-channel attacks. *SideLine* has the advantage of being portable on a wide range of devices as it does not require the presence of specific circuitry (e.g. FPGA). Because *SideLine* feeds upon SoC complexity, we also believe that it represents a major threat for actual high-end SoC security. More importantly this threat is likely to scale up in line with the constant performance improvements in SoCs and memory devices.

References

1. Bilal I Abdulrazzaq, Izhal Abdul Halin, Shoji Kawahito, Roslina M Sidek, Suhaidi Shafie, and Nurul Amziah Md. Yunus. A review on high-resolution CMOS delay lines towards sub-picosecond jitter performance. *SpringerPlus*, 2016.
2. Limited ARM. ARM PrimeCell MultiPort Memory Controller (PL176) Technical Reference Manual. Technical report, 2003.
3. Eric Brier, Christophe Clavier, and Francis Olivier. Correlation Power Analysis with a Leakage Model. *Cryptographic Hardware and Embedded Systems*, 2004.
4. Sebanjila Kevin Bukasa, Ronan Lashermes, H el ene Le Boudier, Jean-Louis Lanet, and Axel Legay. How TrustZone Could Be Bypassed: Side-Channel Attacks on a Modern System-on-Chip. In *Lecture Notes in Computer Science*. 2018.
5. Claudio Canella, Daniel Genkin, Lukas Giner, Daniel Gruss, Moritz Lipp, Marina Minkin, Daniel Moghimi, Frank Piessens, Michael Schwarz, Berk Sunar, Jo Van Bulck, and Yuval Yarom. Fallout: Leaking data on meltdown-resistant CPUs. *Proceedings of the ACM Conference on Computer and Communications Security*, 2019.
6. Fei Chen, Yi Shan, Yu Zhang, Yu Wang, Hubertus Franke, Xiaotao Chang, and Kun Wang. Enabling FPGAs in the cloud. In *ACM Computing Frontiers*, 2014.
7. Ching Che Chung, Pao Lung Chen, and Chen Yi Lee. An all-digital delay-locked loop for DDR SDRAM controller applications. In *International Symposium on VLSI Design, Automation and Test*, 2007.
8. Joan Daemen and Vincent Rijmen. *The Rijndael Block Cipher*. 1999.
9. Jean-Max Dutertre, Bruno Robisson, Assia Tria, and Loic Zussa. Investigation of timing constraints violation as a fault injection means. *Design of Circuits and Integrated Systems*, 2012.
10. Dennis R. E. Gnad, Jonas Krautter, and Mehdi B. Tahoori. Leaky Noise : New Side-Channel Attack Vectors in Mixed-Signal IoT Devices. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019.
11. Joseph Gravellier, Jean-max Dutertre, Philippe Loubet Moundi, Yannick Teglia, and Francis Olivier. Remote Side-Channel Attacks on Heterogeneous SoC. *18th Smart Card Research and Advanced Application Conference*, 2019.
12. Daniel Gruss, Cl ementine Maurice, and Stefan Mangard. Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript. *Detection of Intrusions and Malware, and Vulnerability Assessment*, jul 2015.
13. Zijo Kenjar, Tommaso Frassetto, David Gens, Michael Franz, and Ahmad-Reza Sadeghi. VOLTpwn: Attacking x86 Processor Integrity from Software. *CoRR*, 2019.
14. Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pages 361–372, jun 2014.
15. Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre Attacks: Exploiting Speculative Execution. In *2019 IEEE Symposium on Security and Privacy (SP)*, may 2019.
16. Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. *Advances in Cryptology*, 1999.

17. Anil Kurmus, Nikolas Ioannou, Matthias Neugschwandtner, Nikolaos Papandreou, and Thomas Parnell. From random block corruption to privilege escalation: A filesystem attack vector for rowhammer-like attacks. *11th USENIX Workshop on Offensive Technologies*, 2017.
18. Moritz Lipp, Andreas Kogler, David Oswald, Michael Schwarz, Catherine Eason, Claudio Canella, and Daniel Gruss. PLATYPUS: Software-based Power Side-Channel Attacks on x86. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021.
19. Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown. *CoRR*, jan 2018.
20. ST Microelectronics. STM32MP1 Reference manual. 2019.
21. K Murdock, D Oswald, F D Garcia, J Van Bulck, D Gruss, and Frank Piessens. Plundervolt: Software-based Fault Injection Attacks against Intel SGX. *41st IEEE Symposium on Security and Privacy*, 2020.
22. Colin O’Flynn and Alex Dewar. On-Device Power Analysis Across Hardware Security Domains.: Stop Hitting Yourself. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019.
23. OpenSSL Software Foundation. <https://www.openssl.org/>, 2002.
24. Pengfei Qiu, Dongsheng Wang, Yongqiang Lyu, and Gang Qu. Voltjockey: Breaching trustzone by software-controlled voltage manipulation over multi-core frequencies. *Proceedings of the ACM Conference on Computer and Communications Security*, 2019.
25. Joaquin Romo. DDR Memories Comparison and overview.
26. Falk Schellenberg, Dennis R. E. Gnad, Amir Moradi, and Mehdi B. Tahoori. An inside job: Remote power analysis attacks on FPGAs. In *Design, Automation & Test in Europe Conference & Exhibition*, 2018.
27. Falk Schellenberg, Dennis R. E. Gnad, Amir Moradi, and Mehdi B. Tahoori. Remote inter-chip power analysis side-channel attacks at board-level. In *Proceedings of the International Conference on Computer-Aided Design*, New York, NY, USA, nov 2018. ACM.
28. Adrian Tang, Simha Sethumadhavan, and Salvatore Stolfo. CLKSCREW: Exposing the Perils of Security-Oblivious Energy Management. In *26th USENIX Security Symposium (USENIX Security 17)*, 2017.
29. Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient out-of-Order Execution. In *Proceedings of the 27th USENIX Conference on Security Symposium, SEC’18, USA*, 2018. USENIX Association.
30. S van Schaik, A Milburn, S Österlund, P Frigo, G Maisuradze, K Razavi, H Bos, and C Giuffrida. RIDL: Rogue In-Flight Data Load. In *2019 IEEE Symposium on Security and Privacy (SP)*, 2019.
31. Colin D Walter. Simple Power Analysis of Unified Code for ECC Double and Add. In *Cryptographic Hardware and Embedded Systems*, 2004.
32. Zane Weissman, Thore Tiemann, Daniel Moghimi, Evan Custodio, Thomas Eisenbarth, and Berk Sunar. JackHammer: Efficient Rowhammer on Heterogeneous FPGA-CPU Platforms. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, dec 2020.
33. Xilinx. Zynq-7000 SoC Data Sheet. 190:1–25, 2012.
34. Lu Zhang, Luis Vega Gutierrez, and Michael Bedford Taylor. Power Side Channels in Security ICs: Hardware Countermeasures. *CoRR*, 2016.

35. Mark Zhao and G. Edward Suh. FPGA-Based Remote Power Side-Channel Attacks. In *IEEE Symposium on Security and Privacy*, 2018.
36. Yongbin Zhou and DengGuo Feng. Side-Channel Attacks: Ten Years After Its Publication and the Impacts on Cryptographic Module Security Testing. *IACR Cryptology ePrint Archive*, 2005.
37. Kenneth M. Zick and John P. Hayes. Low-cost sensing with ring oscillator arrays for healthier reconfigurable systems. *ACM Transactions on Reconfigurable Technology and Systems*, 2012.
38. Kenneth M. Zick, Meeta Srivastav, Wei Zhang, and Matthew French. Sensing nanosecond-scale voltage attacks and natural transients in FPGAs. *ACM/SIGDA*, 2013.

9 Appendix

Algorithm 1 Zynq processor attack, AP#0 attack pseudo-algorithm

Input: Nb_{acq} , Nb_{sample}
 DMA_{init}();
 UART_{init}();
while Nb_{acq} has not been reached **do**
 Send AES plaintext to AP#1;
 Launch DMA transfer(Nb_{sample});
 Send $Start_{AES}$ to AP#1;
 Wait for End_{AES} flag();
 Wait for End_{DMA} flag();
 Export samples through UART;
end while

Algorithm 2 Zynq processor attack, AP#1 victim pseudo-algorithm

Input: AES_{key} , $AES_{plaintext}$
 AES_{init}();
while infinity **do**
 Wait for $Start_{AES}$ flag();
 Get AP#0 plaintext;
 OpenSSL AES encrypt();
 Send End_{AES} flag to AP#0;
 Send AES ciphertext to AP#0;
end while

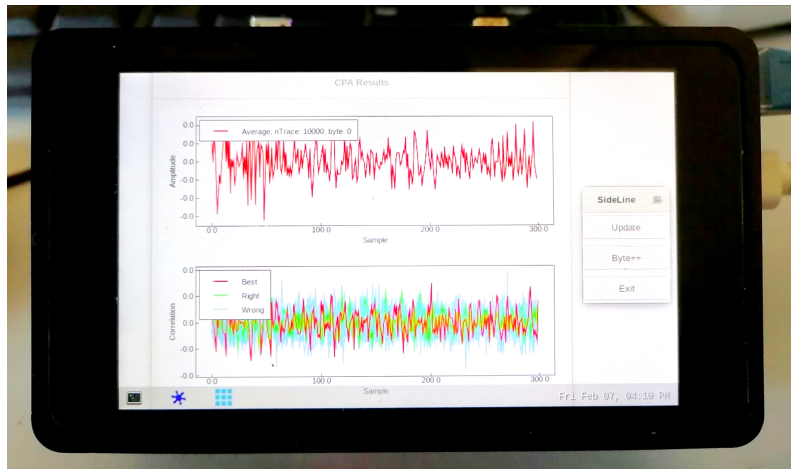


Fig. 11: AES traces acquisition, CPA computation and GTK display (implemented for demonstration) are all embedded in the same application running within the STM32MP157-DK2 board.

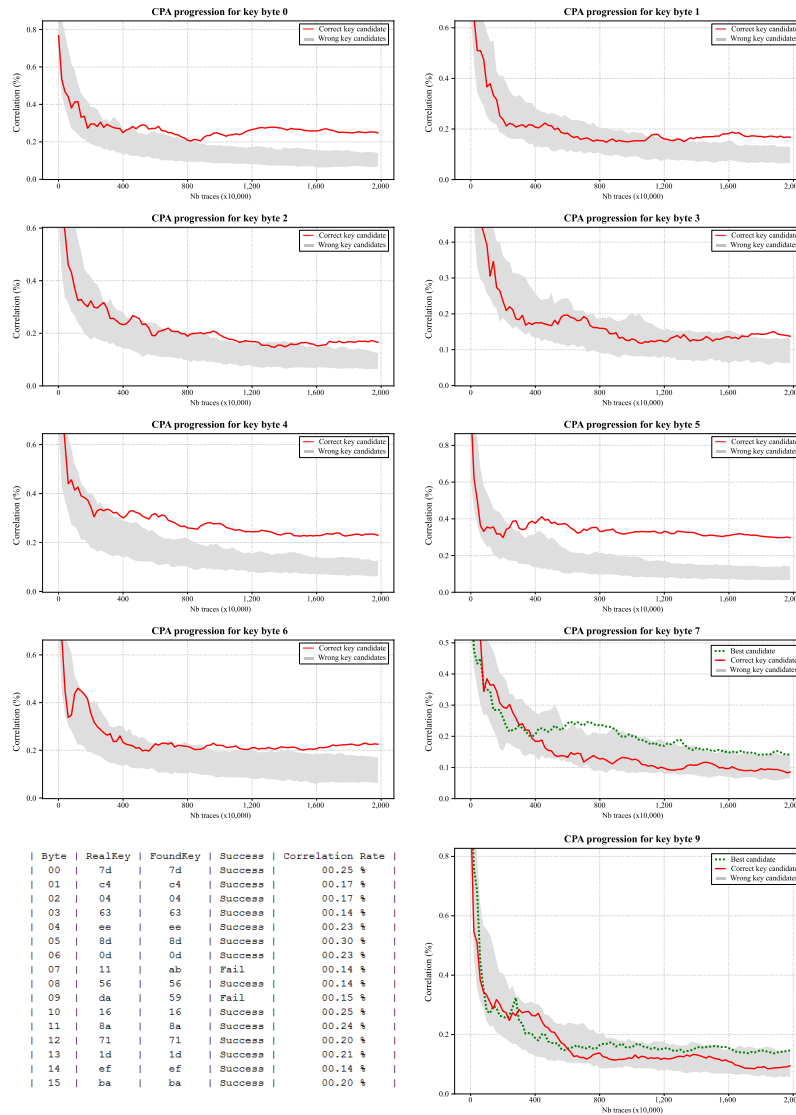


Fig. 12: **ZYNQ AP-vs-AP attack scenario** - The CPA progression (y-axis) over the number of traces (x-axis) is represented for the first 8 AES key bytes. Bytes 7th and 9th which never emerged from the incorrect key candidates are also represented. These CPA results were obtained over 20 million AES encryptions, the correlation rates are provided in the summary table.

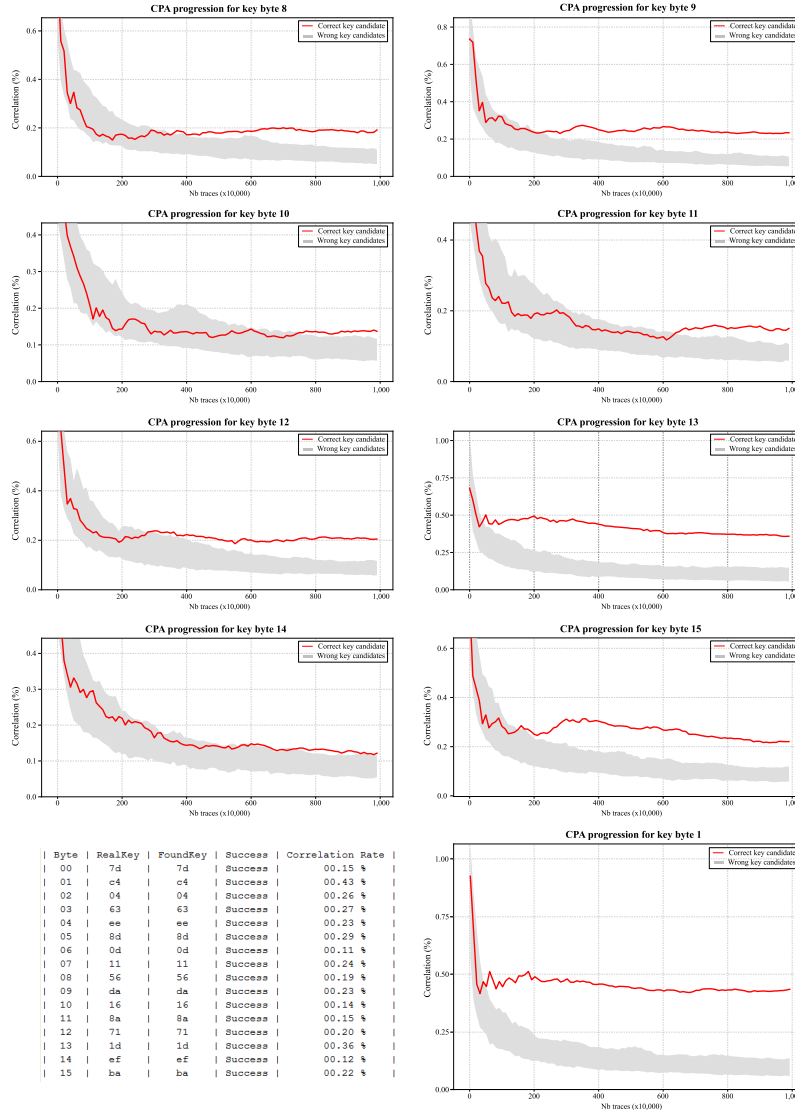


Fig. 13: **STM32MP1 AP-vs-MCU attack scenario** - The CPA progression (y-axis) over the number of traces (x-axis) is represented for the last 8 AES key bytes. The 1st AES key byte is also represented as it provided the best correlation rate. These CPA results were obtained over 10 million AES encryptions, the correlation rates are provided in the summary table.

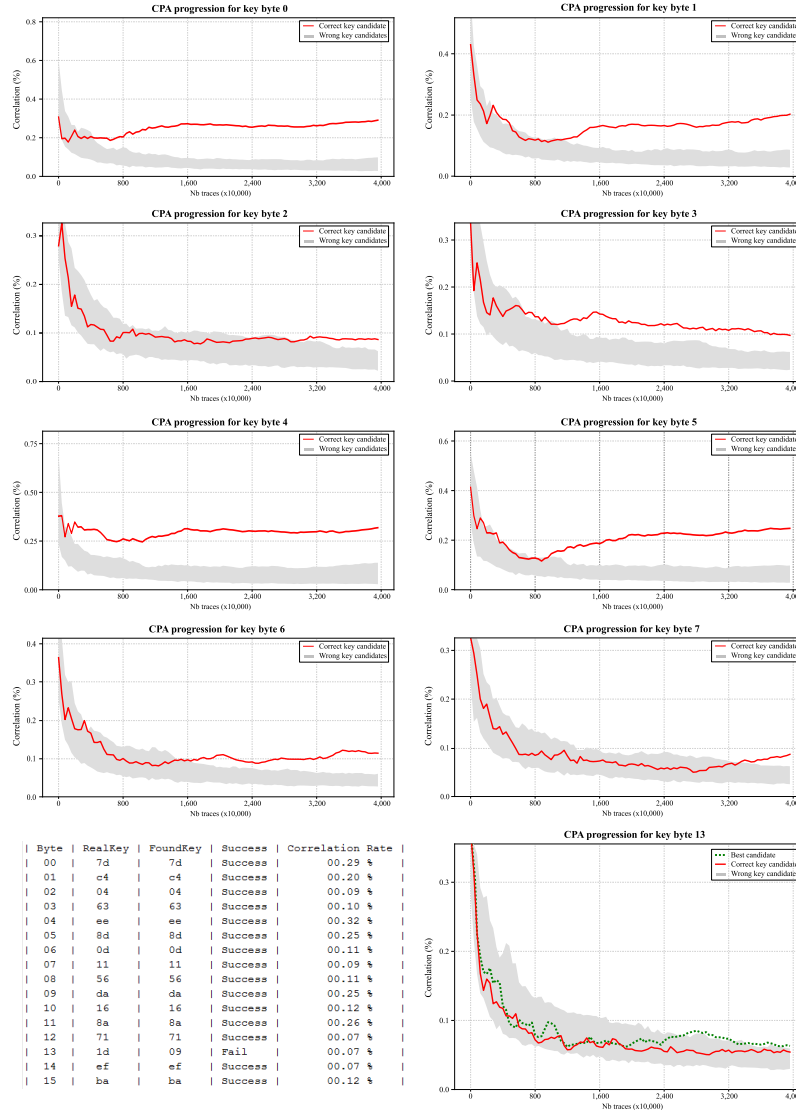
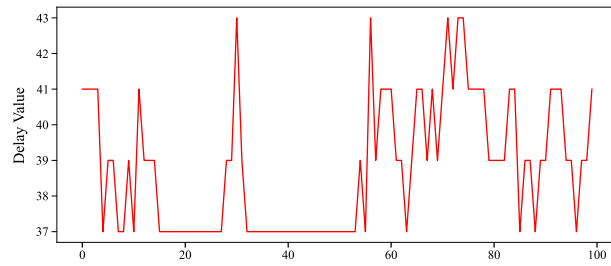
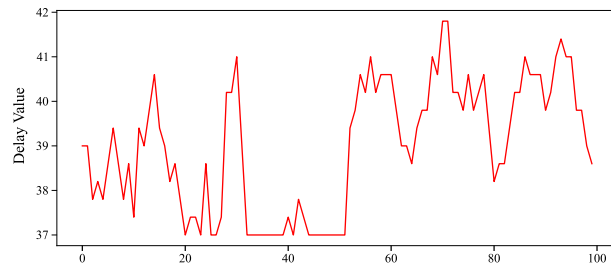


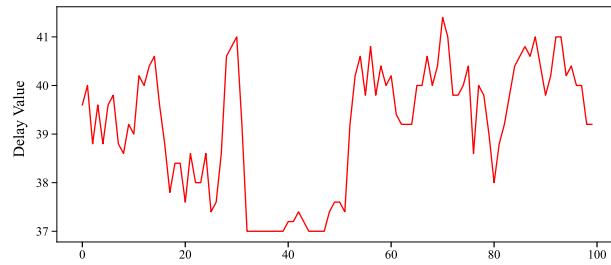
Fig. 14: **STM32MP1 MCU-vs-AP attack scenario** - The CPA progression (y-axis) over the number of traces (x-axis) is represented for the first 8 AES key bytes. Bytes 13th which never emerged from the incorrect key candidates is also represented. These CPA results were obtained over 40 million AES encryptions, the correlation rates are provided in the summary table.



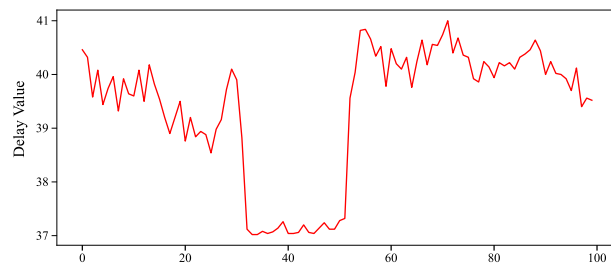
a) Delay value obtained for a 1 AES encryption



b) Averaged delay value obtained for 5 AES encryptions



c) Averaged delay value obtained for 10 AES encryptions



d) Averaged delay value obtained for 100 AES encryptions

Fig. 15: **STM32MP1 MCU-vs-AP attack** scenario: This figure illustrates the delay-block resolution limitation when a single AES encryption is acquired (a). This resolution can be virtually increased by averaging a higher number of traces: 5 (b), 10 (c) and 100 (d) traces.