Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier xx.xxxx/ACCESS.xxxx.DOI

DFAulted: Analyzing and Exploiting CPU Software Faults Caused by FPGA-Driven Undervolting Attacks

Dina G. Mahmoud[1](Member, IEEE), David Dervishi[1], Samah Hussein[1], Vincent Lenders[2](Member, IEEE) and Mirjana Stojilović[1](Senior Member, IEEE)

[1]School of Computer and Communication Sciences, EPFL, Lausanne, Switzerland (e-mail: dina.mahmoud@epfl.ch, david.dervishi@epfl.ch, samah.husseinyoussef@epfl.ch, mirjana.stojilovic@epfl.ch)

[2]Cyber-Defence Campus, armasuisse, Thun, Switzerland (e-mail: vincent.lenders@armasuisse.ch)

Corresponding author: Dina G. Mahmoud (e-mail: dina.mahmoud@epfl.ch).

This research is supported by armasuisse Science and Technology.

ABSTRACT

Field-programmable gate arrays (FPGAs) combine hardware reconfigurability with a high degree of parallelism. Consequently, FPGAs offer performance gains and power savings for many applications. A recent trend has been to leverage the hardware versatility of FPGAs with the software programmability of central processing units (CPUs) to improve the performance of processing-intensive workloads. A variety of heterogeneous FPGA-CPU embedded systems are thus available. However, the security of FPGA-CPU systems has not yet been thoroughly evaluated. In this work, we demonstrate the first attack on FPGA-CPU platforms which leverages undervolting caused by the FPGA to inject faults and exploit them against a software encryption algorithm. The aggressor FPGA affects a CPU sharing the same system-on-chip (SoC). We show that circuits in the FPGA fabric, controlled by an attacker, can create a significant supply voltage drop which, in turn, faults the software computation performed by the CPU or even causes a denial-of-service attack. Our results do not rely on any hardware modifications of the target platform. We present a characterization of the attack parameters and the effects observed. Then, we leverage the FPGA-induced undervolting to fault multiplications executing on the CPU. We also highlight how an attacker might benefit from the injected faults to compromise the system's security by demonstrating differential fault analysis (DFA) against an advanced encryption standard (AES) implementation. Our work exposes a new electrical-level threat in tightly integrated modern FPGA-CPU SoCs, bringing to light a need for more research on countermeasures.

INDEX TERMS FPGA, heterogeneous computing, differential fault analysis, remote attacks, undervolting

I. INTRODUCTION

For decades, Moore's law and Dennard's scaling have been steadily driving the magnificent computing performance growth. As Moore had predicted, the number of transistors on integrated circuits (ICs) was doubling every 18 months. Higher on-chip speeds were also possible, for the same power per unit area [1]. However, the power efficiency has come to an end, limiting the benefit of higher transistor density. Therefore, Moore's law is becoming obsolete. To keep up with the increasing computing performance needs, the landscape of computing systems is drastically changing. Heterogeneous systems have enhanced the traditional, central processing unit (CPU) based platforms. In heterogeneous systems, different types of processing units, each well adapted to a range of user workloads, are combined.

Field-programmable gate arrays (FPGAs), thanks to their bit-level programmability, can be used to build custom hardware accelerators. FPGAs are well suited for a variety of applications, e.g., high-performance computing, machine learning, industrial, automotive, aerospace, and defense. In heterogeneous systems, FPGAs can be incorporated as standalone circuits or as part of a system-on-a-chip (SoC) together with a general-purpose CPU. Both Xilinx-AMD and Intel [2], [3] offer FPGA SoCs. FPGAs can also be combined with CPUs even if not on the same chip, which is often the case in datacenters and the cloud [4]–[6].

With the inclusion of FPGAs in the cloud came the increasing interest in their security. Researchers have discovered that malicious users can build FPGA circuits capable of various attacks [7]. For instance, logic elements in the programmable logic (PL) of the FPGA can be programmed to measure on-chip circuit delays, which are affected by the power consumption. The measured delays can, in turn, reveal secret information about the accelerators deployed on the same chip. Both side-channel and covert-channel vulnerabilities, which can affect commercial cloud platforms, have been demonstrated [8]–[10]. FPGAs can host what are known as *power-wasting circuits* and draw significant current. As a consequence, the on-chip voltage is lowered, affecting the delays of other circuits sharing the same power supply. Researchers have shown that such voltage variations, if left unmonitored, can cause denial-of-service (DoS) [11]–[13]. Finally, with careful control, adversaries can leverage the voltage fluctuations for more subtle fault-injection exploits [14]–[16].

CPUs are not immune to the security issues either. While CPU vulnerabilities can arise from many sources, recent research has focused on the risks associated with remote access to the power management interfaces [17]–[22]. It has been shown that remote monitoring can lead to side-channel exploits [21]. Furthermore, access to *dynamic voltage and frequency scaling* (DVFS) interfaces can be exploited for fault injection via overclocking [22] or undervolting [17], [19], [20].

Combining FPGAs and CPUs in the same heterogeneous system creates the possibility of new, unexplored security vulnerabilities. Indeed, in our previous work [23], we have demonstrated that a malicious user, residing on an FPGA, can create voltage disturbances capable of faulting CPU computation. In this manuscript, we extend our previous work by presenting an actual exploit against a CPU encryption code, which leverages FPGA-generated undervolting not only to fault the CPU computation, but also to steal cryptographic secrets. We provide a detailed analysis of the effects of FPGA-based power wasting on the voltage supplied to the FPGA SoC, and examine the factors affecting the success of the attack. Our analysis highlights the various consequences that such an attack can have, which range from no effect to a DoS, passing by fault injection. To the best of our knowledge, we demonstrate the first successful remote differential fault analysis (DFA) exploit against the advanced encryption standard (AES), in which computational faults are injected into the CPU-based AES code by undervolting entirely originating from the FPGA.

In summary, this manuscript extends our previous work [23] as follows:

- We demonstrate an exploit against a realistic CPU victim code by carrying out DFA against an AES code.
- We enhance the FPGA power-wasting circuits implementation to modify the attack effects, and introduce additional attack control parameters. We execute and present a thorough exploration of the new attack circuit and its parameters.
- We compare our findings of the FPGA-based attack against a DVFS-based setup.
- We provide a detailed analysis of the vulnerability of two victim applications, multiplication and AES, to the attack parameters under our control.

The remainder of this paper is organized as follows. Background and related work are given in Section II. The attack threat model is explained in Section III. System design and the experimental evaluation are presented in Sections IV and V, respectively. The results are summarized and discussed in Section VI. Section VII concludes the paper.

II. BACKGROUND AND RELATED WORK

In this section, we discuss how remote exploits based on electrical-level vulnerabilities are launched on CPUs, FPGAs, and heterogeneous systems. In particular, we highlight attack mechanisms as well as the basics of exploiting injected faults. We also present an overview of recent works which have leveraged such attacks.

A. TIMING FAULTS

Electrical-level fault-injection exploits rely on inducing timing faults in the target victim circuit. Let us look at the diagram in Fig. 1, showing a combinational circuit, whose inputs are supplied from registers and whose outputs are saved in registers. If the circuit is operating at a clock period of T_{clk}, then the timing faults occur if the following inequality is *not* satisfied:

$$T_{\mathrm{clk}} \geq T_{\mathrm{clk2q}} + T_{\mathrm{setup}} + T_{\mathrm{crit}} - T_{\mathrm{skew}} \qquad (1)$$

Here, T_{clk2q} is the time between the clock edge and the corresponding update of a flip-flop's output, which translates to the time needed by the input register to supply the new value to the combinational circuit. T_{setup} is the time for which the value at the input of the output register must remain stable, before the clock edge arrives. T_{crit} is the longest delay of all paths through the combinational logic (i.e., the critical path delay) and T_{skew} is the delay caused by different clock arrival times to the input and output registers [24].

To violate the circuit's timing constraints, the adversary can either manipulate the clock signal (increase its frequency,

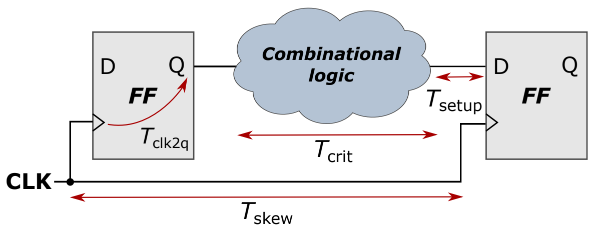


FIGURE 1: Circuit timing parameters, which together form constraints that must be respected for correct operation.

i.e., overclock the design), or manipulate the voltage supply. Lowering the supply voltage results in increased circuit delays, potentially making the sum at the right hand of the inequality higher than the clock period.

B. REMOTE TIMING-FAULT ATTACKS ON CPUS

DVFS is a technique for reducing the circuit's power consumption by dynamically adjusting the supply voltage or the operating frequency. The technique exploits the linear and quadratic relationships between the power and the operating frequency and voltage, respectively. DVFS is available in many commodity processors [25]. Moreover, DVFS interfaces are software-accessible, though a certain privilege level is required to access them.

The software accessibility of DVFS interfaces makes modern CPUs potential victims of remote timing fault-injection attacks. Tang et al. were the first to exploit the DVFS interfaces on ARM processors; they overclocked the processor well past its operating limits and, consequently, observed faults [22]. Other researchers opted for undervolting as the fault-injection mechanism [17]–[20]. Successful attacks against Intel and ARM processors, targeting the trusted execution environments (TEEs) such as ARM TrustZone and Intel SGX, were demonstrated. The exploits included breaking the security of encryption algorithms and violating memory safety [17]–[20].

C. REMOTE TIMING-FAULT ATTACKS ON FPGAS

Heterogeneous FPGA SoCs, similarly to CPUs, may sometimes have DVFS interfaces [27]. So far, research on remote fault-injection attacks on FPGAs did not make use of DVFS; rather, it focused on a different form of attacks, in which the voltage is manipulated indirectly—with the help of the FPGA logic. As mentioned in Section I, FPGA resources can be used to build power-wasting circuits, capable of lowering the voltage of the entire chip. With the lowered voltage, circuit delays increase and, if the conditions are right, a fault can be injected or the host platform can be reset. A number of power-wasting circuits have been explored in the literature. Designs may use combinational ring oscillators (ROs) (an inverter in a closed loop), as shown in Fig. 2a [11]. Alternative designs introduce a flip-flop or a latch within the feedback loop of the ROs, as shown in Fig. 2b [26]. Power-wasting circuits may also employ glitch generators and amplifiers, similar to the design shown in Fig. 2c. Researchers have even proposed designs using block random access memories (BRAMs), and overclocked AES encryption rounds [12], [26], [28], [29].

Power-wasting FPGA designs typically incorporate an enable signal, controlled by the attacker. For a combinational RO, adding an enable signal typically translates to implementing a NAND functionality in a look-up table (LUT) and driving one of its inputs by the LUT output, as illustrated in Fig. 2a. For other power wasters, the control changes, e.g., one may need to manipulate the signals on the clear or preset ports of flip-flops, to ensure high-enough switching frequency and increased power consumption. By carefully controlling the enable signal of its logic and, consequently, the obtained voltage waveform, attackers can recover secret AES keys, activate stealthy hardware Trojans, and bias random number generators and neural networks [14]–[16], [30]. Previously demonstrated attacks had one thing in common: both the attacker and the victim resided on the same FPGA.

D. HETEROGENEOUS SYSTEMS EXPLOITS

While the focus has been mostly on attacking individual computing units, recent attention has been directed towards heterogeneous computing systems. For instance, Weissman et al. induced RowHammer bitflips in the CPU's main memory, using an FPGA-based attacker [31]. Many other works focused on gaining information through side channels or establishing channels for covert communication. Giechaskiel et al. demonstrated covert channels between graphics processing units (GPUs), CPUs, and FPGAs, sharing the same power supply unit in a datacenter [9]. They made use of ROs on the FPGA to act as both transmitters and receivers, since the RO's oscillation frequency is affected by the supply voltage. For the CPU and the GPU, they used computationally heavy codes to transmit messages. Leveraging sensors on the FPGA, Zhao et al. demonstrated the feasibility of side-channel attacks against the CPU on the same chip [10]. Finally, Gnad et al. showed that the high power consumption and the subsequent resetting that occurs due to the ROs activity not only affects the FPGA fabric, but also affects a CPU on the same chip [11]. Our work shows the possibility of FPGA-initiated undervolting affecting a CPU on the same chip, and injecting faults that can facilitate exploits such as DFA against AES.

E. DIFFERENTIAL FAULT ANALYSIS AGAINST AES

AES is the algorithm of choice for symmetric encryption (encryption and decryption performed using the same secret key). It can be incorporated into modern devices as either a crypto-accelerator or as a software crypto-library [32]. AES is a block-cipher algorithm operating on 128-bit blocks. It uses a key whose length can be 128, 192, or 256 bits. AES consists of applying round transformations to the input multiple times, where the number of rounds depends on the size of the key used. For a 128-bit key, AES requires ten rounds to encrypt the input plaintext. The rounds consist of the following operations: AddRoundKey, MixColumns (not performed in the last round), SubBytes, and ShiftRows, as shown in Fig. 3 [33].

Due to its widespread use, AES is one of the most common attack targets. Side channels and injected faults can both be used to recover the AES secret key. For fault injection-based exploits, DFA is among the techniques commonly used.

In DFA exploits, the attacker relies on being able to send the same plaintext twice to be encrypted by the victim. One of those encryptions occurs under normal operating conditions, and is therefore, correct. The other occurs while the attack is active. The timing of the fault injection needs to be carefully

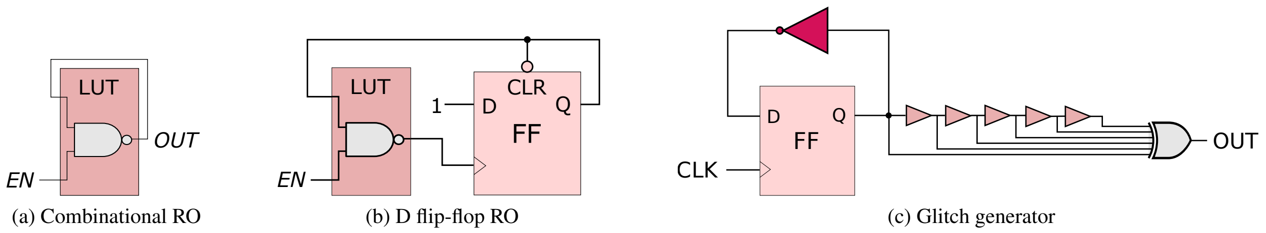


FIGURE 2: Examples of power wasting circuits: (a) combinational RO, (b) a register-based power waster, suitable for exploits in cloud FPGAs [26], and (c) another variation of power wasters combining registers with glitch generators [12].

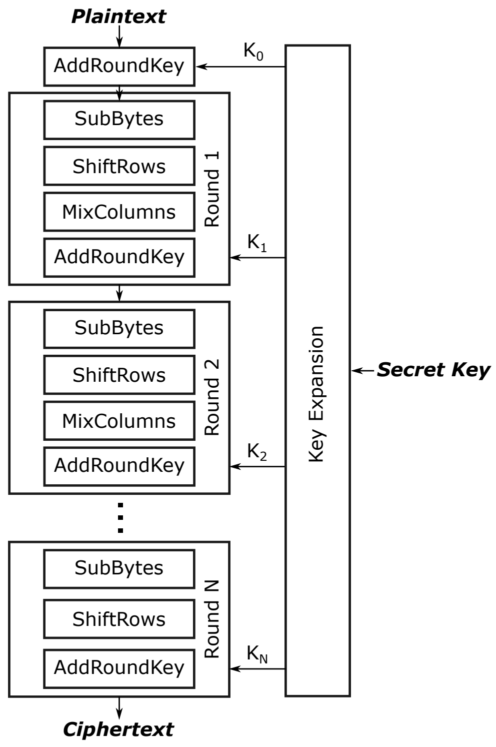


FIGURE 3: Illustration of the AES algorithm.

controlled. A fault injected too early or too late will result in a faulty ciphertext that is not exploitable for DFA. If the adversary times the fault injection to occur before the penultimate round (for a key size of 128 bits, after the eighth round and before the ninth round), a single injected fault will result in four faulty bytes in the ciphertext [33]. If a fault is injected before the eighth round, then all of the ciphertext bytes will be faulty, which can allow the recovery of the key using a single fault [32]. Depending of the number of pairs of ciphertexts collected, the key recovery may or may not require some brute forcing.

III. THREAT MODEL

Our work focuses on heterogeneous systems combining CPUs and FPGAs. For voltage variations to propagate across components of the system, the power delivery network (PDN) needs to be shared. This sharing may occur at a variety of levels (e.g., silicon die, board, datacenter rack, etc.). In our case, we consider a system-on-chip platform, where the FPGA and the CPU are within the same package [2], [3] and share the on-chip PDN. Furthermore, the SoC is powered from the common voltage source, as shown in Fig. 4; such a setting is common in various commercially available platforms [34], [35].

We consider a remote exploit scenario; therefore, the adversary is limited to software access to the target platform. On the CPU side, this means that the attacker can access the CPU and run some code on one of its cores (attacker core in Fig. 4). On the FPGA side, the adversary instantiates power-wasting circuits to lower the voltage of the system. Consequently, the attacker needs to be able to upload the (partial) bitstream to the FPGA (i.e., have privileges to program the FPGA for some time). Alternatively, the adversary needs to be able to communicate with the circuits on the FPGA to control an already existing power-wasting hardware Trojan. While the control of the attack can be done from the CPU, and thus requires communication with the FPGA, this feature is not necessary and is part of our experimental setup only for demonstration purposes.

We make no assumptions regarding the availability of DVFS interfaces on the target platform, as our attack does not make use of them. In the experimental analysis, we use the DVFS interfaces to compare the effects of different frequencies and voltage levels on the CPU operation. However, the adversary is not assumed to have access to DVFS interfaces. If DVFS interfaces exist on the target SoC, then our threat model assumes that the victim is free to use them for performance or power saving reasons.

The victim runs its application on one of the available cores of the processing system (PS) (victim core in Fig. 4). The other cores in the system can be attacker-controlled, victim-controlled, or neither. They can also be idle or busy.

The attacker can have multiple goals with the undervolting attack; a DoS exploit is a possibility. An alternative is fault injection. In particular, the injected faults can lead to DFA exploits against encryption algorithms, such as AES. For a successful DFA attack, we make the assumption that it is possible for the attacker to send some data to the victim and to observe the corresponding output, sent over a public channel. This assumption is in line with the classic DFA

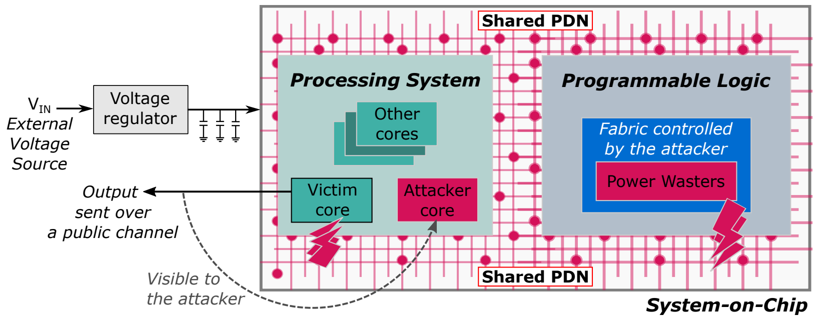


FIGURE 4: Threat model, where the attacker-controlled programmable logic induces faults in the victim core, through the shared power delivery network. The victim can send outputs on a publicly observable channel, which enables DFA exploits against some encryption algorithms.

threat models [16], [22], [36].

In our threat model, the adversary can only indirectly attempt to disturb the voltage of the CPU cores, by generating voltage fluctuations in the part of the chip where the reconfigurable fabric is.

IV. SYSTEM DESIGN

We test the undervolting-based attack on the Genesys-ZU board [34]. The board features one Zynq UltraScale+ MPSoC (XCZU3EG). The MPSoC makes use of a quad-core ARM Cortex-A53 as an application processing unit (APU) (which we sometimes refer to as CPU), a dual-core ARM Cortex-R5F as a real-time processing unit (RPU), and a Mali-400 MP2 GPU, which is not used in our experimental evaluation. Unlike voltage, the frequency of the APU can be controlled from software. The PS and the PL share common power supply voltage regulators.

To be able to compare our results against previous work, which relied on the availability of DVFS interfaces, we take one AMD-Xilinx ZCU102 development board; it features the same MPSoC as the Genesys-ZU (with a larger PL). The DVFS interfaces of ZCU102 board allow us to control the voltage and frequency of the APU [27]. In the remainder of this section, we explain the setup for using the DVFS. We then present the attacker and the victim designs.

A. DVFS SETUP

We make use of the DVFS setup with the goal of exploring the voltage and frequency operating limits of the ARM Cortex-A53 CPU. The ZCU102 development board allows changing both the voltage and the frequency of the CPU, thanks to the on-board Maxim InTune digital point-of-load (POL) controller and the Maxim PowerTool software on the host PC (i.e., the computer communicating with the board). The command `VOUT_COMMAND` allows setting the CPU operating voltage according to the following formula [37]:

$$V_{\mathrm{CPU}} = V \times 2^{-12}, \tag{2}$$

where V_{CPU} represents the voltage controller output voltage and V is the value written by the command. Accordingly, one can sweep the voltage with a granularity of 0.244 mV. In our experiments, we first send the command to set the voltage to a low level (i.e., the *undervolting* command) and immediately after a command to restore the voltage to its nominal level. Hence, the resulting voltage impulse lasts for a short time to emulate the impulse an attacker would generate from the programmable logic (which has an attack duration of 2.56 μs). It is worth noting that the level of the timing control from the host PC is rather limited, resulting in less accurate attack timing than when launching the attack from the PL.

The frequency control mechanism is the same for both Genesys-ZU and ZCU102. From software, we can write values to the register `APLL_CTRL` to change the feedback divider. The clock frequency is calculated as follows [38]:

$$F_{\mathrm{CPU}} = \frac{\mathrm{PS\_REF\_CLK} \times \mathrm{FBDIV}}{\mathrm{DIV} + 1} \tag{3}$$

where F_{CPU} is the output frequency of the APU, PS_REF_CLK is the default reference clock for the processing system (33.33 MHz on the ZCU102 and 30 MHz on the Genesys-ZU), FBDIV is a 7-bit value controlling the feedback divider, and DIV is either zero or one, and indicates whether the division by two is enabled in the PLL or not. We can control FBDIV and DIV from software to sweep the frequencies in steps of 15 MHz or 30 MHz for the Genesys-ZU.

B. PROCESSING SYSTEM

The processing system, in our design and threat model, is home to both the attacker and the victim. All of our experiments have no operating system involved, and run bare-metal. Since the effects induced by the adversary affect the entire chip, there is a chance that the attacker application itself will be affected by the undervolting. Therefore, we decide to place the attacker on one of the RPU cores, which run at a lower frequency than the APU cores. The default frequency for the RPU is 500 MHz and can go as high as 600 MHz, while the default frequency for the APU is 1.2 GHz and the maximum recommended frequency is 1.5 GHz [27], [34]. The RPU and the APU clocks are fed

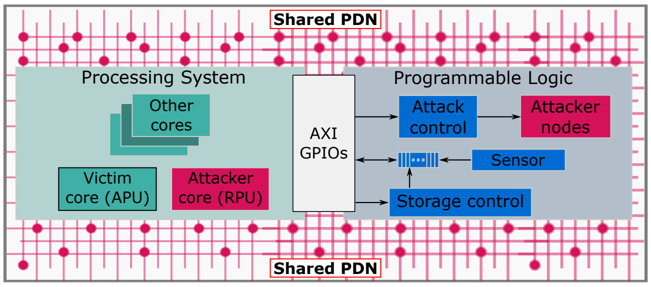


FIGURE 5: System Design.

```
initialize_variables();
expected = calculate_expected_result();
wait(); //wait for the attack to start
while (count <= attack_duration) {
  result = victim_function();
  if (result != expected) {
    store_fault();
  }
  count++;
}
print(stored_faults);
```

Listing 1: Code executing on the victim core.

by separate PLLs; hence, the changes to the APU frequency do not affect the frequency of the RPU. We note here that the choice to run the attacker on the RPU reduces the likelihood of the attacker faulting, but is not a requirement of the exploit. If the target system does not have an RPU, the attacker can reside on one of the other cores and accept the lower success rate of the attack caused by errors in the attacker core. In fact, during some of our experiments, the attacker faults, stopping the exploit and invalidating the collected data point.

In our test scenario, the processing system has three parts, as shown in Fig. 5. The first part is the victim core, responsible for setting the APU frequency. The target code to attack is also executed here. For testing purposes, the code begins by initializing the variables and calculating the expected result. The calculation and storage of the expected result is done prior to the start of the attack to avoid the value being affected. Then, the victim function is repeated within a loop for the duration of the attack. Whenever the output is not as expected, it is stored to be analyzed after the attack. We structure this loop to carry out only the victim function and the error information storage, to aim for the attack only affecting the victim. Once the exploit is over, the victim outputs whether a fault has occurred or not. If a fault has occurred, it also reports the fault so that one can analyze it. This way, post-processing can help ascertain that reported faults are indeed faults, and can report the type of fault, and whether or not it is exploitable. The victim algorithm is presented in Listing 1. The structure of the code is similar to the proof-of-concept used in previous research [19].

Then, we have the attacker core, which sets up the communication with the FPGA and controls the attack. The adversary can configure all of the programmable parameters of the attacker circuit in the PL. The attacker core is also the one that controls the start of the exploit. The code starts by initializing the variables and registers for communication with the FPGA. Our attacker code repeats the exploit for a certain number of times, with the attack parameters specified by the host PC. We control all of the programmable parameters and change them according to the type of the experiment. While the attack is running, the attacker core waits until the end to avoid its computation being affected by the exploit. After the end of all the attack runs, the attacker core outputs the voltage sensor readings, recorded in the programmable logic during the experiment. Voltage sensor readings are only collected to analyze the effects of the undervolting.

Finally, we have the remaining APU cores, which are neither victim-controlled nor attacker-controlled. These cores do not influence the attack in any way. In some of our experiments, we will vary the choice of the APU core for the victim code for exploration and analysis. Additionally, we will make the remaining cores either idle (i.e., not running any code) or busy. The code keeping the cores busy will typically be similar to the victim code, even though the adversary will not care for potential faults induced in any but the victim core.

C. PROGRAMMABLE LOGIC

According to our threat model, the power supply voltage fluctuations originate from the programmable logic, i.e., from the FPGA. Hence, in our attack setup, we let the attacker have unconstrained access to the PL. This is an acceptable assumption, because the FPGA does not need to be only spatially shared between applications; it can also be temporally shared (i.e., the adversary could be given full control over the FPGA for a limited time).

To generate voltage fluctuations within the PL, the adversary deploys power-wasting circuits. A variety of FPGA power viruses have been proposed in literature and tested on commercial cloud platforms [26], [28], [29]. While all the designs can be deployed in one or more commercial cloud platforms, some cloud service providers run stricter checks than others and prevent the users from deploying potentially dangerous circuits (e.g., Amazon AWS prevents designs with combinational loops, but Alibaba Cloud still allows them). Given that our experimental setup is not limited by the cloud service provider policies, we design and implement power-wasting circuits which are, first, efficient in generating voltage fluctuations and, second, allow a high degree of control over the timing and the effect of the attack.

We implement enhanced ring oscillators (EROs), the FPGA power wasters previously proposed by La et al. [29]. One instance of an ERO is shown in Fig. 6; it is a collection of four look-up tables (LUTs) implementing NAND functionality. The enable signal of the ERO is driving one input of each LUT. Two LUT inputs are connected to the output of the same LUT, forming a combinational loop. The remaining three inputs are driven by the outputs of the other three LUTs in the ERO. Compared to the traditional combinational ROs,

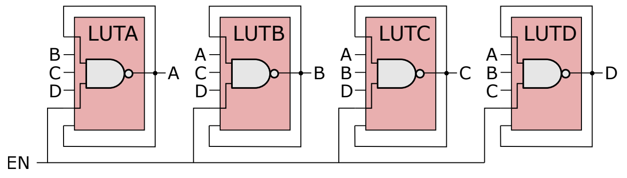


FIGURE 6: The schematic of an enhanced ring oscillator (ERO). Letters A, B, C, and D refer to the outputs of the four LUTs, which are also driving the inputs of other LUTs.

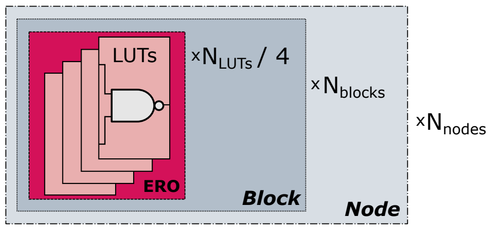


FIGURE 7: Attacker organization.

LUTs in the ERO drive higher capacitive load, resulting in increased overall power consumption [29]. In Section V, we will experimentally compare EROs and ROs (the power-wasting circuits we used in our previous work [23]).

To allow fine-tuning of the attack, we instantiate EROs (respectively, ROs) in groups of N_{LUTs} look-up tables in total, which we term *blocks*. Knowing that one ERO instance (respectively, RO) cannot create a significant disturbance on its own, we treat one entire block of EROs as the smallest unit of control. We then group N_{blocks} into *nodes*, with each node having its own activation signal, and instantiate N_{nodes} nodes in total. This hierarchical structure of clustering the power wasters is illustrated in Fig. 7.

Besides the flexibility of setting the number of active nodes and the active blocks per node, we implement a mechanism for fine-tuning the timing of the attack. In particular, the following parameters can be set, as shown in Fig. 8:

- the start of the attack, i.e., the moment when the first attacker node is enabled,
- the period of the enable signal, i.e., the number of clock cycles between two subsequent activations of the enable signal,
- the duty cycle of the enable signal, i.e., the number of clock cycles during which the enable signal remains high over the period of the enable signal, and
- the duration of the attack.

To maximize the effect of the attack [23], we activate (and deactivate) the nodes in a *staggered* manner (i.e., we enable (disable) one additional node at each subsequent clock cycle). To control all the parameters, we use advanced extensible interface (AXI) general-purpose input/outputs (GPIOs), as shown in Fig. 5.

The last and optional component we implement in the PL is a voltage sensor. It allows us to observe the effects of the attack parameters on the resulting voltage disturbance. Once

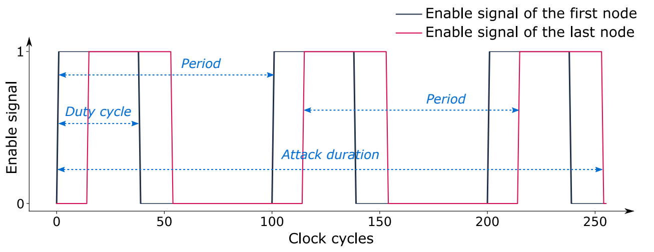


FIGURE 8: Enable signals for the first and the last-activated attacker nodes, illustrating the periodic activation pattern and the duty cycle of the enable signal.

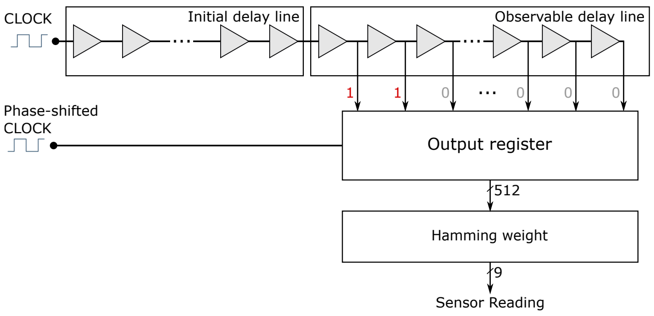


FIGURE 9: TDC sensor for measuring programmable logic delay changes caused by the supply voltage fluctuations.

the attack parameters are chosen, the sensor readings can be ignored. However, the sensor may be helpful for side-channel analysis, e.g., to decide when to trigger the attack [39]. We implemented a time-to-digital converter (TDC), as shown in Fig. 9, similarly to previous research on FPGA-based power side-channel and fault-injection attacks [8], [14], [30], [39]–[41]. The TDC sensor consists of carry-chain propagation elements, forming a delay line. The clock signal enters and propagates through the delay line. The state of the delay line is captured in the output register, clocked with the phase-shifted clock of the same frequency. The phase shift between the two clocks is set during the calibration, which we performed before running the experiments. The sensor reading is a numerical value corresponding to the number of carry-chain elements (the buffers in the observable delay line in Fig. 9) through which the clock signal has propagated by the time the observable delay line state is captured in the output register. In our TDC implementation, the *sensor reading* is simply the Hamming weight of the output register. Since the supply voltage affects the logic propagation delay, the sensor reading provides information regarding the delay change, and accordingly the voltage change. The relationship between the sensor reading and the voltage is approximately linear, as shown in the comprehensive study by Moini et al. [42].

V. EXPERIMENTAL EVALUATION

In this section, we present the results of the analyses of the vulnerability of the processor inside an FPGA SoC to the FPGA-based undervolting attacks. Firstly, we find the CPU operating limits (the voltage and frequency) in the absence of an attack. Then, we focus on the FPGA-based attacker and analyze the effects of the attack parameters on the success of the attack. We proceed with presenting the results of the fault-injection attacks against two victims: a multiplication code and an AES code. Finally, we demonstrate a successful DFA attack against the CPU while it is performing AES encryption.

A. OPERATING LIMITS

Understanding the operating limits without the attack is essential for creating realistic attack scenarios. For instance, it would not make sense to try to attack a victim operating at a clock frequency that would fault its operation even in the absence of an attack.

The frequency operating limits are straightforward to test on both Genesys-ZU and ZCU102 platforms. We sweep the frequency using the `APLL_CTRL` register and Equation (3). For the Genesys-ZU board, we use steps of 15 MHz for most of the evaluation; we use the step size of 30 MHz (by changing the DIV parameter to 0) only to validate the CPU operation at higher frequencies than those achieved with a DIV parameter of 1 (the maximum programmable frequency for a DIV parameter of 1 is 1.905 GHz, while it is 3.81 GHz when DIV is 0).

We collect the operating frequency limits for several software applications, to account for the varying signal paths in the CPU. The investigated victim applications are:

- multiplication (Appendix B),
- TinyAES (Appendix C),
- addition (Appendix D),
- subtraction (Appendix D),
- loads and stores from memory (Appendix E), and
- printing to the standard output (Appendix F).

All applications run within a loop, where we try to catch any faults that occurred. Moreover, we keep track of the faults which result in the CPU aborting the operation, and monitor the communication between the processor and the host CPU. Finally, we execute the tests once with all the remaining APU cores busy running the same code as the victim. After identifying the first APU core which faults in the previous test (i.e., the *busy* tests), we let only that core run the victim code while we keep the remaining APU cores in the idle state. These experiments allow us to obtain the range of safe operating frequencies for the victim.

For all the tested configurations, we find that the operating frequency can be set well past the maximum recommended frequency of 1.5 GHz, without any problems occurring during the code execution. In particular, we find that 1.89 GHz is the maximum safe operating frequency for all the tested applications, regardless of the state of other APU cores on the Genesys-ZU board. The results are summarized in Table 1. Similarly, for the ZCU102 board, we find that the maximum safe operating frequency is around 2 GHz, as shown in Table 2. While the processor is the same on both boards, there are some differences, including the PDN and the clock

TABLE 1: Maximum safe operating frequency at the nominal voltage (0.85 V) for the Genesys-ZU, once with other APU cores idle and once with other APU cores busy.

Application	First Core[1]	Idle (MHz)	Busy (MHz)
Multiplication	0	1890	1890
TinyAES	0	1890	1890
Addition	1	1905	1905
Subtraction	2, 3	1920	1905
Load & Store	1	1905	1905
Printing	3	1920	1920

TABLE 2: Maximum safe operating frequency at the nominal voltage (0.85 V) for the ZCU102, once with other APU cores idle and once with other APU cores busy.

Application	First Core[1]	Idle (MHz)	Busy (MHz)
Multiplication	0, 2	2033	2000
TinyAES	0, 1, 2	2000	2000
Addition	1, 2, 3	2000	2000
Subtraction	2, 3	2016	2000
Load & Store	1, 2, 3	2033	2000
Printing	2	2016	2000

generator, which could account for the maximum frequency difference. Knowing that the maximum operating frequency is rather conservative, the victim may choose to overclock its core (e.g., for improved performance). Hence, in our attack exploration experiments, we sweep APU operating frequencies both below and above the maximum recommended frequency.

Thanks to the availability of the DVFS interface on the ZCU102 platform, it is straightforward to obtain the voltage operating limits. In the case of the Genesys-ZU, one can read voltage from the hardened on-chip voltage sensors while the attack is running and correlate them with the effects observed in software. However, as the voltage fluctuates during the attack and the on-chip voltage sensors have a relatively limited sampling frequency, the results may be inaccurate. As both experimental platforms use the same FPGA SoC, we evaluate and report here the voltage operating limits obtained with ZCU102.

Fig. 10a shows the maximum and minimum voltage at which we observe faults for the multiplication victim at each frequency point. The solid lines correspond to the case when other cores are idle, while the dashed lines correspond to the case when other cores are busy. For each frequency point, the maximum faulting voltage is the highest voltage at which we observe a fault. The minimum voltage is the point immediately before we start observing the loss of communication with the core. For the reported faulting voltage range, we repeated tests several times, with the same and with different multiplication operands. We show a sample of the operands

[1]First core refers to the ID of the first APU core that faults when the CPU operating frequency is further increased.

TABLE 3: Locations of multiplication faults (bit flips) for several operand values.

A	B	Faults
0xdeadbee	0x445566778	0x0000000000010000
0x445566778	0xdeadbee	0x0000000000c00000
0xff5566778	0xdeadbee	0x0000000000c00000
0xdeadbee	0xff5566778	0x0000000000010000
0xB7A7D0	0x21E4D	0x0000000180000000
0x10	0x32AC87D99	0xff00000000000000

tested and the observed faults in Table 3. The faults are the result of XORing the expected result with the obtained result to show the faulty bits. We found that the possibility of inducing a fault mainly depended on the voltage and frequency pairs, while the exact fault observed depended on the operands.

We repeat the same analysis for TinyAES, the AES code we will analyze in further detail in Section V-E. The results are shown in Fig. 10b. As expected, we observe a trend similar to the multiplication case: lower voltage is required to fault or crash the application at a lower working frequency. However, the exact values are not the same for TinyAES and for multiplication. This mismatch is not surprising, as different pieces of code use different data and control paths in the CPU. As we will soon see, the observed differences correspond well to the results of the FPGA-based undervolting attacks. Finally, in line with previous work [18], the other cores being busy increases the maximum voltage at which faults (and crashes) start occurring. However, the trend remains similar to the idle case.

B. ATTACKER PARAMETERS SWEEP

The adversary's aim is to induce a controlled voltage drop, which lasts long enough to propagate to and fault the CPU, but not long enough to reset the board. CPU-based attacks have control over the exact voltage values because they rely on DVFS interfaces [17], [19]. However, they lack the precise timing enabled by the hardware design deployed on an FPGA [43]. As a result, they can maintain a specific voltage value for a significant amount of time, resulting in a fault, but they are also more likely to inject multiple faults, because of the reduced control over the attack duration.

Our attack, on the other hand, allows for precise control over the duration of the attack. We set the duration in the hardware as a specific number of clock cycles (for a fixed clock frequency of 100 MHz) for which the attack will last. We also periodically activate and deactivate the attacker resulting in the lowest voltage drop lasting for a few specific clock cycles within the attack duration. However, because of the use of power-wasting circuits and their periodic activation, we do not have the same level of control over the voltage value. This being said, a fixed number of power-wasting circuits with a specific activation pattern will produce approximately the same minimum voltage value and undervolting pattern every time. Therefore, our exploit is also predictable once the mapping between the attacker parameters and the voltage is known.

TABLE 4: Resource utilization by the RO-based attacker.

Unit	LUTs Total	LUTs in Percentage	FFs Total	FFs in Percentage
Block	343	0.49%	0	0%
Node	4629	6.56%	44	0.03%
Total	60963	86.4%	660	0.47%

TABLE 5: Resource utilization by the ERO-based attacker.

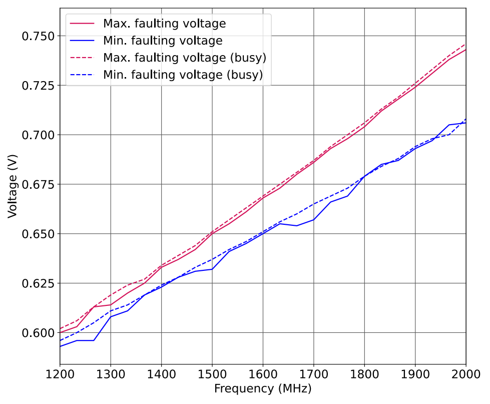
Unit	LUTs Total	LUTs in Percentage	FFs Total	FFs in Percentage
ERO	4	<0.01%	0	0%
Block	124	0.18%	0	0%
Node	3932	5.57%	59	0.04%
Total	58981	83.59%	885	0.63%

In our previous work [23], for the proof-of-concept of the attack, we used combinational ROs. For a more effective attack and a wider exploration, we here change the adversary's circuits to EROs (Section IV-C), and increase the level of control we have over the attack. We repeat the attack parameters sweep and explore the additional attack control parameters with EROs as the power wasters. The resource usage is reported in Tables 4 and 5.

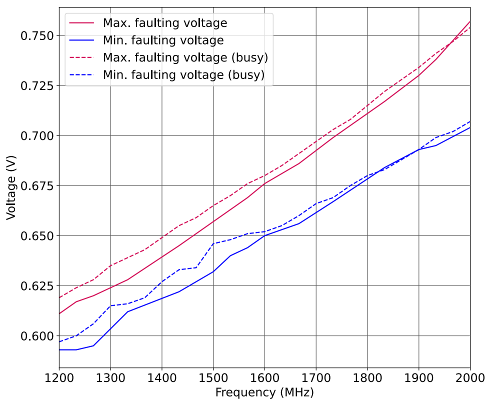
The aim of our first experiment is to compare the voltage drops generated using ROs and EROs. It is expected that due to the extra power consumption in the routing of the EROs, we will observe a more significant voltage drop. The results are shown in Fig. 11 for an attack duration of 256 clock cycles (2.56 µs). The baseline corresponds to the sensor readings recorded in the absence of an attack. Even though the resource utilization of the EROs is smaller than that of the ROs, EROs are indeed more effective at creating the voltage disturbances. We also note here that when the EROs are deactivated, we observe some overvolting due to the voltage regulator attempting to quickly recover the nominal voltage.

In Fig. 12, we show the voltage drop in the function of the number of ERO nodes and blocks per node. We observe that the attacker configurations with the same total number of blocks result in a slightly different voltage drop. For example, 10 nodes of 30 blocks each and 15 nodes of 20 blocks each have the same total number of blocks (300 blocks). However, the resulting voltage shape varies because the staggered activation of the nodes takes longer when there are more nodes to enable. Regarding the lowest value of the observed voltage, we find that it is mostly affected by the total number of active blocks; the higher it is, the lower the voltage.

Regarding the period of the enable signal and the duty cycle, we observe the same effects when using ROs and EROs, as we show in Fig. 13. Specifically, if we increase the duty cycle beyond 50%, the board is more likely to



(a) Multiplication



(b) TinyAES

FIGURE 10: Comparison of the voltage operating limits of APU core 0 (ZCU102) in the function of the CPU frequency, when other cores are idle (full line) or busy (dashed line), for the multiplication and TinyAES, respectively.

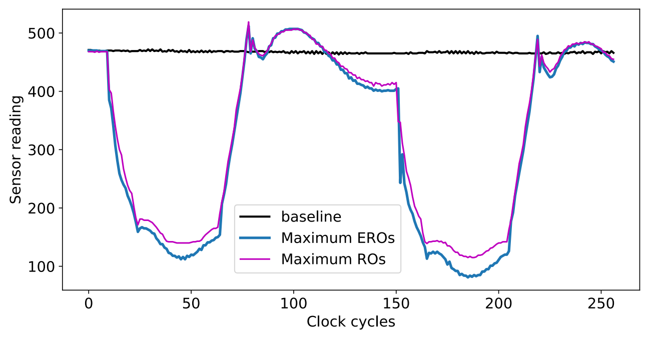


FIGURE 11: Comparison of the voltage drop induced by the maximum number of EROs and the voltage drop induced by the maximum number of ROs for the same activation parameters.

reset; accordingly, we restrict the duty cycle to below 50%. If the activation period is too short, the voltage drop is not significant because the power wasters are not enabled for a sufficiently long time for the voltage to drop. If the activation period is too long with respect to the attack duration, we observe one voltage drop pulse during the attack duration. Furthermore, that voltage drop is not more significant than those observed with smaller periods. Hence, for a specific range of activation periods (neither too long nor too short), we can observe the most effective voltage drop and also generate more than one disturbance per attack duration.

Next, we show that staggering the activation (and deactivation) of the nodes is a more effective strategy than activating them all simultaneously [23]. In Fig. 14, we see that longer-lasting and thus more effective voltage drop is indeed obtained; such a voltage shape more closely resembles the voltage an adversary with physical access to the device or to the DVFS interfaces would be able to obtain. Furthermore, the staggered (de)activation allows the attacker

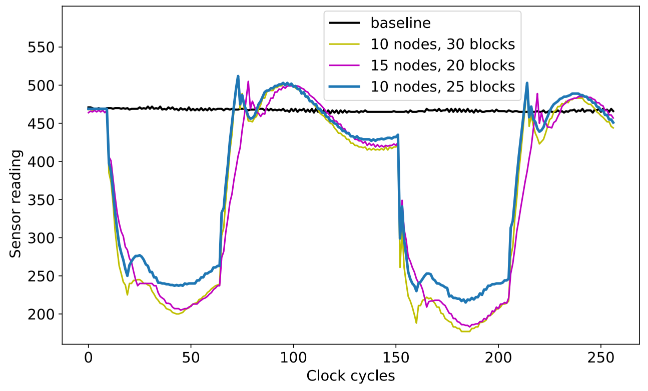


FIGURE 12: Effects of the number of active nodes and the total number of active blocks on the resulting voltage drop. The higher the total number of blocks, the more the voltage drops. Due to the staggered activation of the nodes, an attack with a larger number of nodes takes longer time to drop the voltage.

to avoid extreme overshoots and undershoots of the voltage, which are not useful for the exploit because of their short duration.

Finally, we record the values of the voltage as reported by the on-chip system monitor, and list a sample of these values in Table 6. We note here that the analog-to-digital converter used in the system monitor operates at 0.2 MHz [44], a lower rate compared to the TDC voltage sensor we employ to observe the voltage fluctuations (100 MHz). Hence, we take the readings from the system monitor only as an indication of the voltage values that occur, rather than the accurate measure of the minimum voltage. The minimum value may be skipped due to the reduced sampling speed of the ADC with respect to the TDC, which we observe in Table 6 as the voltage does not always decrease when the total number of active blocks grows. However, using the TDC readings (e.g., in Fig. 12),

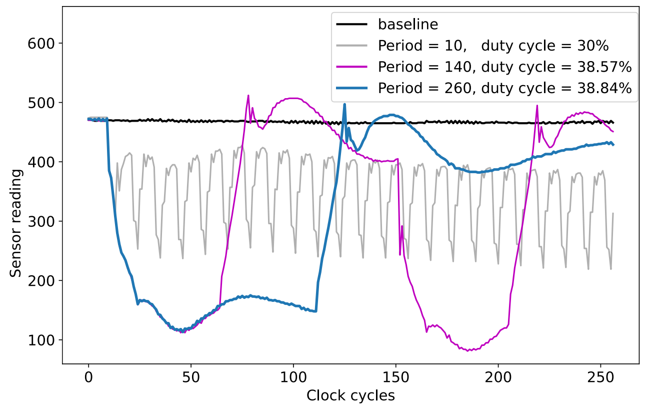


FIGURE 13: Comparison of the voltage drop induced by different periods (in clock cycles) of the attacker enable signal for a maximum size of the ERO-based attacker.

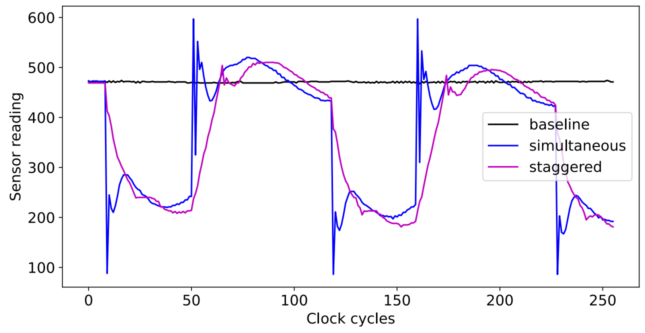


FIGURE 14: Voltage drop induced by activating 15 of the ERO-based attacker nodes simultaneously or in a staggered fashion for a block size of 20.

TABLE 6: Minimum voltage values reported by the system monitor for a varying number of active ERO blocks.

Nodes	Blocks per Node	Total Blocks	Voltage in the PS (V)	Voltage in the PL (V)
9	22	198	0.738	0.712
11	25	275	0.715	0.674
15	25	375	0.688	0.639
15	26	390	0.680	0.636
15	27	405	0.683	0.633
15	29	435	0.688	0.639
15	30	450	0.662	0.627

we see that the minimum voltage value is lower for a bigger attacker size. From Table 6, we also see that the PL voltage is always lower than the PS voltage; this is expected because the PL, being the source of the attack, is also the component most affected by it.

Table 7 summarizes the range of all the investigated attack parameters and presents the parameters we have chosen for the attacks discussed in the following sections.

C. FAULT INJECTION AGAINST MULTIPLICATION

We choose to test our undervolting attack against what we deem an interesting subset of applications of the ones tested in Section V-A. We first test our fault injection capabilities against a proof-of-concept multiplication code (Appendix B) similar to the one used in Plundervolt [19]. Multiplication is a widely-used operation in many software algorithms, and faulting it can allow for exploits, such as violating memory safety, and redirecting a victim to an attacker-controlled part of the memory [19]. We enhance the victim code functionality to allow recording multiple faults if they occur, instead of stopping as soon as a fault has occurred (Listing 1).

Multiplication is implemented as a multiply-add instruction on the ARM Cortex-A53 on which we run our experiments. We sweep the chosen attack parameters and the APU frequency for a specific pair of operands to gain an understanding of when correct operation is expected, when faults can be injected, and when denial-of-service occurs. We show the results of this sweep for one pair of operands (`0x445566778 and 0xdeadbee`), in Figs. 15 and 16, for when other cores are idle and busy, respectively. We note here that the DoS points are due both to the board resetting and to the loss of communication with the victim core. A reset requires that we power cycle the device in order to continue the experiment. On the other hand, loss of communication is usually recoverable by restarting the code on the processor from the Xilinx Software Development Kit (SDK).

We record two types of faults: the faults in the multiplication result and synchronous aborts. While the default behavior when a synchronous abort occurs is for the code to enter an infinite loop, we consider it an injected fault instead of a DoS attack (although an attacker aiming for DoS may make use of synchronous aborts to achieve that). Synchronous aborts can occur due to a variety of reasons, but all of them indicate that a fault has occurred [45]. We interpret this as fault injection being possible, and observe that changing the timing of the attack with respect to the victim can result in a faulty computation instead of a synchronous abort (as it will be shown in the analysis we do for DFA on AES). Finally, in Figs. 15 and 16, normal operation refers to cases where the victim code executes and terminates correctly.

The trends observed in Figs. 15 and 16 are similar between the two figures and when compared to using DVFS (Fig. 10). The faults start occurring at a lower frequency for the busy cores case, again matching the results in Fig. 10. Finally, in the cores busy case, we observe a more frequent occurrence of the faults than the DoS, for the same attacker size and frequency pairs, because the noise introduced by the busy cores makes the effects of the attack less deterministic.

D. FAULT INJECTION AGAINST TINYAES

An attacker trying to compromise the security of a system can easily do so by retrieving the encryption key used for protecting the output of the victim. Therefore, we choose an encryption algorithm as the victim for the exploit demon-

TABLE 7: Targeted attack parameters.

	N_{nodes}	N_{blocks}	Duration (clock cycles)	Period (clock cycles)	Duty Cycle	Activation	Clock Frequency
Sweep	0–15	0–31	128–16384	10–2200	10–50%	Simultaneous or staggered	100 MHz
Chosen	**10–15**	**25–31**	**256**	**80–110**	**39%**	**Staggered**	**100 MHz**

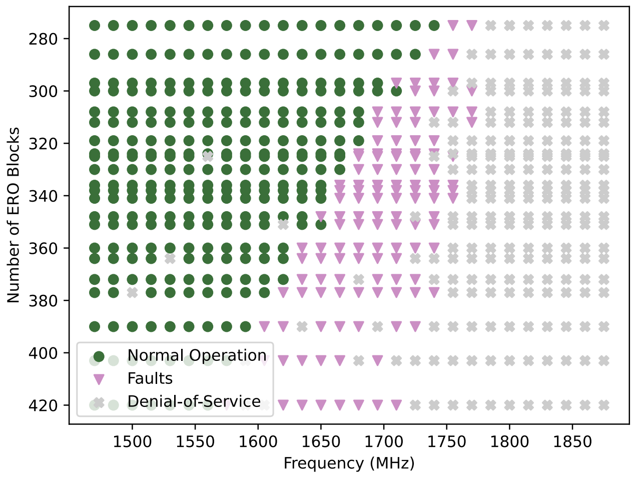


FIGURE 15: Observed effects for various frequency and attacker size pairs, when APU core 2 is running the multiplication code and the remaining APU cores are idle.

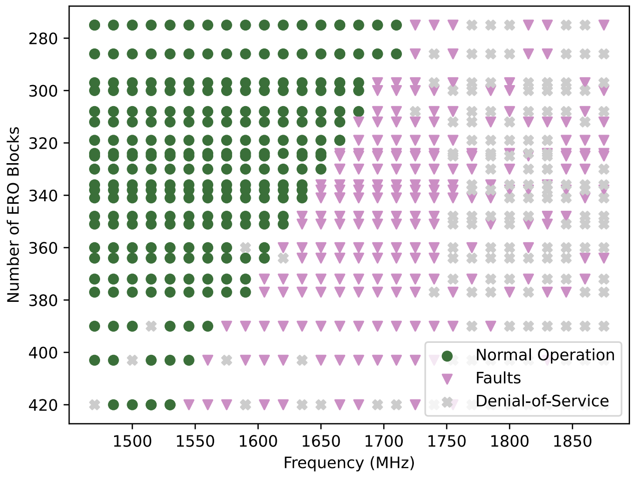


FIGURE 16: Observed effects for various frequency and attacker size pairs, when APU core 2 is running the multiplication code and the remaining APU cores are busy.

stration. We select AES, which is among the widely used encryption algorithms, with high security. Specifically, we use the TinyAES implementation[2] [46], [47], and structure the code as shown in Appendix C.

To explore the vulnerability of TinyAES to undervolting-based fault-injection attacks, we sweep the following parameters:

- the operating frequency of the APU core on which TinyAES is running,
- whether the other cores are idle or busy,
- the number of ERO nodes, and
- the number of ERO blocks per node.

For each set of parameters we record whether no fault happened, a random fault was injected, or a DFA-suitable fault was detected. We also distinguish between synchronous aborts and reset. The results are summarized in Figs. 17 and 18, for one specific APU core, when other cores are idle and busy, respectively. We also show results for higher frequencies, with smaller attacker sizes, for the case when other cores are idle in Fig. 19. Similarly to the proof-of-concept attack on multiplication, we consider a random fault, a DFA-suitable fault, and a synchronous abort as faults, because changing the attack timing can lead to the observation of a different type of fault. We further analyze the effects of the attack timing on the observed faults in the following section. Figs. 17, 18, and 19 show similar trends to those obtained with DVFS (Fig 10b). This similarity is expected as increasing the attacker size results in a more pronounced voltage drop. Additionally, with the cores busy, we see that the faults are more likely, which is consistent with the previous work's observation that having other cores busy reduces the frequency and/or voltage drop for which a fault may occur [18].

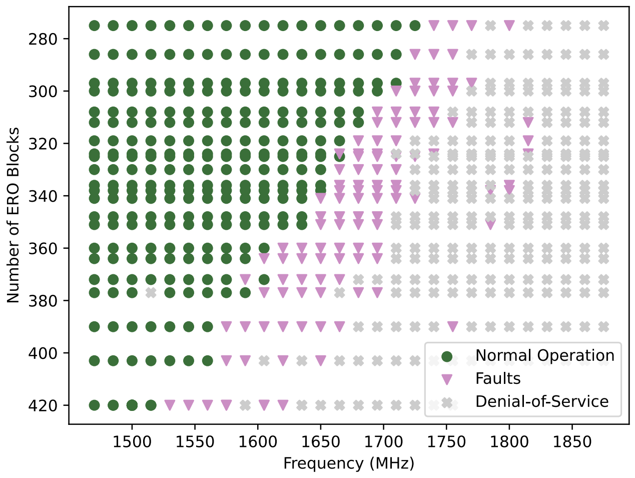


FIGURE 17: Faults observed for various frequency and attacker-size pairs, when APU core 0 is running TinyAES and all other cores are idle.

[2]https://github.com/kokke/tiny-AES-c

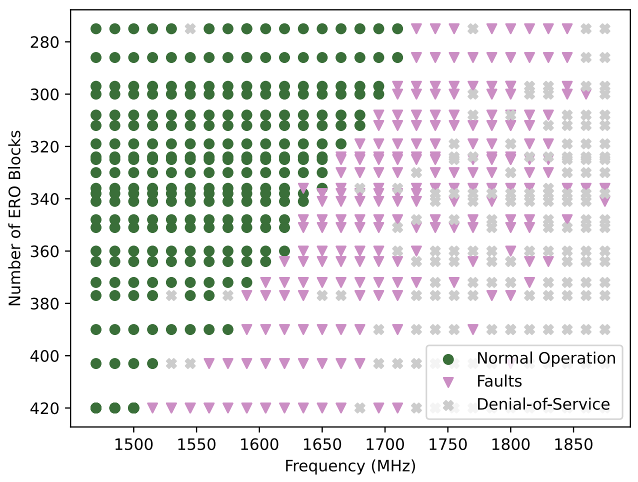


FIGURE 18: Faults observed for various frequency and attacker-size pairs, when APU core 0 is running TinyAES and all other cores are busy.

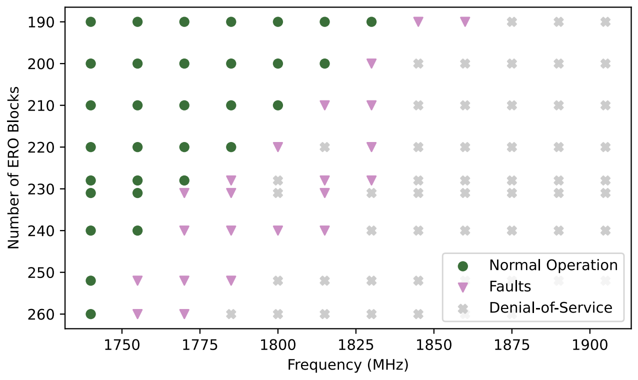


FIGURE 19: Faults observed at higher frequencies with smaller attacker sizes compared to Fig. 17, when APU core 0 is running TinyAES and all other cores are idle. This figure highlights how changing the attacker size allows for injecting faults at more extreme frequencies.

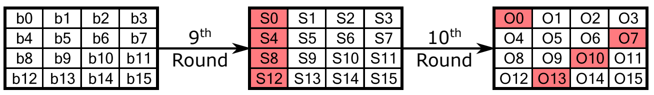


FIGURE 20: Fault propagation to the AES output when one single byte is faulted at the input of the ninth round.

E. DFA AGAINST TINYAES

For the exploit against AES, we aim for a DFA attack targeting the input to the ninth round. The success of such a fault injection is relatively simple for the attacker to validate, as the output will have four faulty bytes, when a single byte has been faulted, as shown in Fig. 20.

For our attack, we assume that the adversary sends a group of sixteen plaintexts to be encrypted by the victim AES. First, the attacker obtains the correct ciphertexts corresponding to these plaintexts. Then, the attacker requests the encryption of these plaintexts again, while launching the attack. The adversary repeats the encryption request until obtaining enough faults for DFA, and can then retrieve the key. For fault injection after the eighth round, eight ciphertext pairs are needed to recover the secret key [48]. If fewer pairs are collected, recovering the key is still feasible, but will involve more brute forcing. For example, we manage to recover the key with only four pairs of correct and faulty ciphertexts, and one correct plaintext with its corresponding ciphertext[3]. We provide the list of plaintexts and ciphertexts used in the attack in Table 8.

We begin by examining the results of the fault injection. For any set of attacker parameters which produced a fault, we know that the resulting undervolting is capable of faulting the computation. The difference between a successful DFA-usable fault injection and a random fault injection is mainly due to the timing of the fault injection attack. In our setup, the timing of the fault injection with respect to the victim execution can be controlled by changing the delays in the attacker code. Particularly, we have delays between each run with a different period for the enable signal in the chosen period range (80–110), and between the runs which sweep the frequency of the enable signal. We refer to these delays as *inter-period* and *inter-run* delays, respectively, and show their place in the code in Appendix A.

We focus on TinyAES operating at 1.5 GHz, the *maximum recommended frequency* of the APU. We find that we are able to inject faults at 1.5 GHz if we use an attacker of 15 nodes and 30 blocks per node. When using this one set of parameters and sweeping the delays, we observe various effects. Figs. 21 and 22 show the results of the delay sweep for an APU core running at 1.5 GHz, with other cores idle and busy, respectively. Two datapoints are not available because the attacker code is faulting, rendering the datapoint invalid. As shown in Figs. 21 and 22, the effect observed changes with the delay. It is also worth noting that more ERO blocks are required for a successful attack at 1.5 GHz than at higher frequencies; consequently, it is more likely for the attack to cause reset, resulting in the DoS points in Figs. 21 and 22. Finally, the state of the other cores affects the success of the attack. We observe more DFA-usable faults in the busy case than in the idle case, and at different attacker delays than the idle case.

Once we find the set of parameters which result in DFA-usable faults, we repeat the attack with those parameters multiple times to understand how likely it is for the board to reset, for no fault to occur, for a random fault to be injected, and for a DFA-suitable fault to happen. For example, for the idle case, we choose the following parameters (according to the results in Fig. 21): 15 nodes, 30 blocks per node, 8,998 µs inter-period delay, and 40,002 µs inter-run delay, and repeat the attack 21 times. In these 21 runs, only two data samples are invalid because of the attacker core faulting. Three times, no fault happened. Six times, the device reset. Finally, a

[3]using the DFA code from: https://github.com/arusson/dfa-aes

TABLE 8: Correct and faulty ciphertexts used in the DFA attack. In red, faulty bytes of the ciphertext. In blue, the corresponding correct bytes of the ciphertext. The first line shows the correct plaintext and ciphertext (without a corresponding faulty ciphertext) used when running the DFA code.

Plaintext	Correct Ciphertext	Faulty Ciphertext
0x103760E6**8B**412BAD179B4FDF45249FF6	0x5DEB028908EE525EAD949A93D4BE6499	N/A
0x103760E67B41**22**AD179B4FDF45249FF6	0x7495EB59806168ECFDD2702D9009D995	0x7495EBA180612EECFDC2702DAC09D995
0x10**11**60E67B412BAD179B4FDF45249FF6	0x9F62603AB62F265C05CD1797F29C8BC2	0x9F6A603AB72F265C05CD1749F29C9BC2
0x103760**FF**7B412BAD179B4FDF45249FF6	0x04339C3D81ACB53BF3F8DEE3ADB6D6FD	0xF6339C3D81ACB5F1F3F866E3AD3AD6FD
0x103760E67B412BAD179B4F**DD**45249FF6	0xD4E0B98A57427333F925DB1013262BF0	0xD4E0998A57AE73334D25DB1013262BDF
Key: 0x3C4FCF098815F7ABA6D2AE2816157E2B		

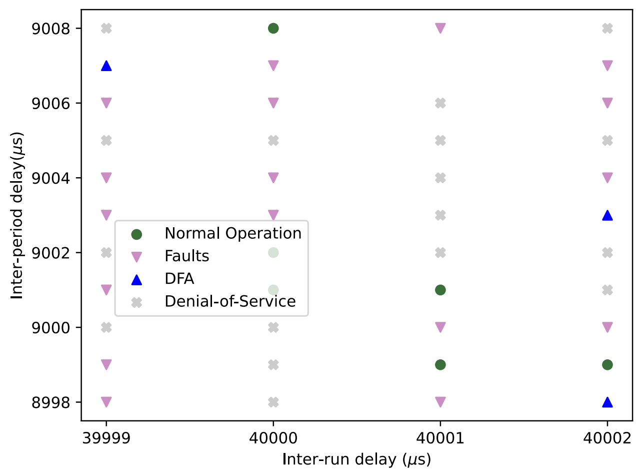


FIGURE 21: Effects observed when changing the inter-period and inter-run delays in the adversary code, with 15 nodes and 30 blocks per node, targeting core 0 operating at 1500 MHz and running TinyAES encryption.

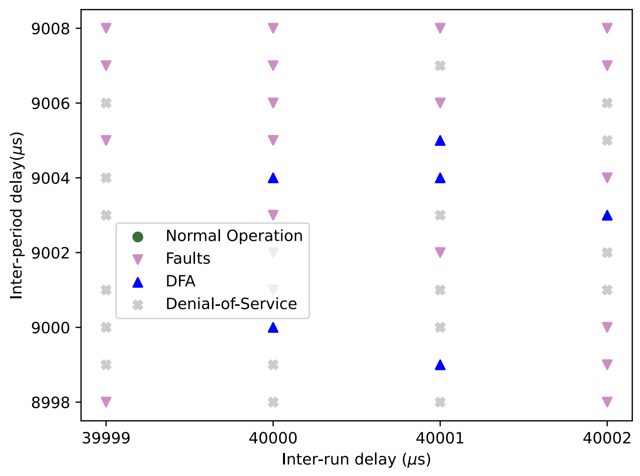


FIGURE 22: Effects observed when changing the inter-period and inter-run delays in the adversary code, with 15 nodes and 30 blocks per node, targeting core 0 in the busy cores scenario, where the cores are operating at 1500 MHz and running TinyAES encryption.

random fault was injected in seven runs out of the 21. A DFA-suitable fault was successfully injected three times (in 14% of the runs). Therefore, we find that good control over the timing of the attack is important for obtaining the DFA-suitable faulty ciphertexts and recovering the encryption key. An example of the collected correct and faulty ciphertext pairs, and the corresponding key is given in Table 8. We note here that the plaintexts in Table 8 are similar to one another; the differences are highlighted in bold.

VI. DISCUSSION

In this work, we have shown how FPGA-based power wasters can be used to inject faults into the computation of a CPU, when the FPGA and the CPU are sharing the same power distribution network. Additionally, we have successfully leveraged the faults for a DFA exploit against an AES code. In this section, we highlight interesting observations derived from our results, discuss future avenues of research, and comment on potential countermeasures.

A. VARYING VULNERABILITIES

Our analysis reveals variations in the extent to which software routines are susceptible to timing faults injection. Starting with the DVFS analysis, we see that the maximum operating frequency of different applications slightly varies, depending on the application itself and on the CPU core on which it is running. Looking more closely at the multiplication and TinyAES, we find that they share a very similar trend of the voltage and the frequency required for the correct operation and for the fault injection. That said, TinyAES seems to be more sensitive to undervolting. When the processor is operating at 1.2 GHz, multiplication faults when the voltage is below 0.6 V; in comparison, TinyAES faults for voltage values below 0.611 V. This difference is not surprising, because every application is a somewhat different sequence of instructions, and instructions may operate on different values and use different hardware paths in the processor. These varying degrees of vulnerabilities are the reason we were able to inject faults into TinyAES at 1.5 GHz, the frequency which is within the recommended operating range. For multiplication, however, we observed faults at higher frequencies only.

Some of the data points in Figs. 15, 16, 17, 18, and 19 do not strictly follow the trends in Fig. 10, because some parts of the attack execution code other than the victim itself are failing. For example, issues with the memory access or with the alignment of the stack pointer and the program counter can lead to synchronous aborts. We have chosen to report such issues as faults for two reasons. First, good timing of the attack is an important factor for the result of the attack and, second, an adversary who knows the vulnerability trend of each victim code can adjust the attack parameters so as to increase the chances of achieving the expected effects (by steering away from configurations in which faults may be due to unrelated processor paths failing).

Overall, our analysis reveals that a successful attack depends both on the attack parameters and the knowledge of the victim code. Since many cryptographic algorithms have well-known code structures and publicly available libraries, adversaries have sufficient resources to help them improve the probability of the attack success.

B. EXTENSION TO OPERATING SYSTEMS

In this work, we considered baremetal code execution. One of the attractive future steps would be to include an operating system (e.g., Petalinux) in the analysis. In that case, many software routines will be executing while the operating system is running. Previously, Murdock et al. [49] reported that kernel panics are one of the recurring effects when lowering the CPU voltage while an operating system is running. To achieve exploitable faults in what is likely to become a much larger attack surface, an FPGA-based attacker may need to further enhance or even develop entirely new attack mechanisms and strategies.

C. SOFTWARE COUNTERMEASURES

The attack's origin in the PL renders some countermeasures, e.g., preventing access to DVFS interfaces or ensuring the integrity of the voltage drivers, ineffective [18]. However, there are other software countermeasures that could protect against the effects of the attack, by detecting the fault injection and ensuring that it cannot be leveraged to break the security of the victim.

Redundancy, for example, is among the common countermeasures against fault injection. Instructions can be executed twice and their results compared, or the code can be executed on multiple threads and the results compared. Any discrepancy in the results would indicate that a problem occurred and, accordingly, that a recovery technique should be applied [50]. Other countermeasures can work at the compiler level to harden the applications, such as Minefield, which places trap instructions that are more likely to fault in the event of undervolting [43]. While the idea of Minefield can be applied to other victims not running in Intel SGX, the presented analysis assumed a DVFS-based attack where the undervolting typically lasts for more than 200 µs. Our attack lasts for a considerably shorter time, meaning that more trap instructions would be required for reliably detecting the attack.

Even though the software countermeasures are useful for protecting sensitive applications, they come at the cost of an increased number of instructions to execute and, accordingly, increased power consumption and latency. They may render the exploits more challenging, but they do not eliminate the root cause of the attack (which, in our case, is the low-level programmability of the FPGA and the shared power distribution network).

D. HARDWARE COUNTERMEASURES

Given that the origin of the undervolting is the hardware of the FPGA, hardware defenses may prove more useful to protect the platform. While hardware countermeasures may incur the cost of changing the hardware, the availability of the PL can be useful for implementing adaptive defenses as the need arises.

Several countermeasures are proposed against FPGA-based exploits. For instance, there is a lot of focus on detecting and forbidding attack circuits, in a similar fashion to Amazon EC2 F1 instances disallowing combinational loops in designs deployed on their FPGAs [13]. Researchers have proposed bitstream scanners to detect malicious primitives in the bitstream and flag them [29], [51]. However, available bitstream scanners target a limited set of commercial FPGAs, and they are not used in all toolchains, so to guarantee protection, the victim would have to incorporate them as a check before uploading any bitstreams to the device. Moreover, researchers have shown that benign-looking circuits, uninteresting to bitstream scanners, can also be turned into effective power wasters [52].

Voltage drop sensors, similar to the one we used, can act both as a malicious primitive for side-channel attacks, and as a facilitator of defense mechanisms. Approaches like those proposed by Provelengios et al. [53] and Mirzargar et al. [54] can locate the source of the voltage drop. Once the source is known, a technique like LoopBreaker [55] can be used to disable the attacker. However, this disabling of the attacker relies on removing its partial bitstream, which takes 1.5 µs to execute. This means that DoS attacks can probably be thwarted, but faults may be injected [55]. While many proposals for countermeasures exist, no single countermeasure can fully protect against the kind of attack we present, and more work is still required to guarantee the security of FPGA-based heterogeneous systems.

VII. CONCLUSION

FPGAs offer performance gains and power savings for a variety of modern workloads. As a consequence, many of today's computing platforms combine the hardware parallelism of FPGAs with the speed and programmability of CPUs. Despite the popularity of FPGA-CPU heterogeneous systems, the security vulnerabilities arising from the tight integration of FPGAs and CPUs remain largely not investigated.

In this work, we presented FPGA-to-CPU undervolting-based exploits. We have shown how, using power-wasting circuits deployed in the FPGA fabric, an adversary can reduce the voltage of the entire platform and, hence, inject faults into the operation of the CPU. We investigated the various factors affecting the strength of the attack and its success. We showed that the injected faults can affect different types of software applications, and can occur within safe operating limits of the CPU. We demonstrated the use of remote FPGA-based undervolting to inject faults for differential fault analysis against AES encryption. Our results present unexplored fault-injection vulnerabilities affecting many heterogeneous platforms, and call for research on effective countermeasures.

.

APPENDIX A ATTACKER CODE

```
int main()
{
    init_platform(); // initialize platform
    init(); // initialize all GPIOs to communicate
     with the FPGA
    initialize_variables();
    calibrate_sensor();
    empty_attack();  // running an attack without
    activating the power-wasters, to record
    baseline sensor values
    wait_for_victim(); // wait for victim code to
    start
    for (int m = 0; m < runs; m++)  // loop for
    the number of targeted attack runs
    {
     period = min_period;  //min period for the
    toggling frequency that has the highest impact
     duty_cycle = period*98/256;
     write_params_to_FPGA();
     while (period <max_period) // max period for
    the toggling frequency that has the highest
    impact
      {
        write_params_to_FPGA();
        start_attack();
        read_data_from_FIFOs();
        usleep(interperiod);
        period = period + 2;  // period increment
        duty_cycle = period*98/256;
      }
     usleep(interruns);
    }
    cleanup_platform();
    return 0;
}
```

APPENDIX B MULTIPLICATION VICTIM CODE

```
int main()
{
    init_platform();
    initialize_variables();
    expected_result=calculate_expected_result();
    set_apu_frequency();
    while (count < attack_duration)
    {
      var = multiplicand; //initialize var with
    multiplicand value
      var *= multiplier; //multiply var with
    multiplier value
      if (var != expected_result)
      {
          record_error();
      }
      count++;
    }
    print_errors();
    cleanup_platform();
    return 0;
}
```

APPENDIX C AES VICTIM CODE

```
int main()
{
    init_platform();
    initialize_variables();
    expected_result=calculate_expected_result();
    set_apu_frequency();
    while (count < attack_duration)
    {
      aes_array = initialize_plaintext_array(); //
    aes_array variable provides the plaintext
    input to the AES encryption and stores the
    ciphertext on returning from the AES function
    call
      AES_encrypt();
      if (memcmp(aes_array, correct_ct) != 0)
      {
          record_error();
      }
      count++;
    }
    print_errors();
    cleanup_platform();
    return 0;
}
```

APPENDIX D ADDITION/SUBTRACTION VICTIM CODE

```
int main()
{
    init_platform();
    initialize_variables();
    set_apu_frequency();
    while (count < attack_duration) //for
    subtraction while(count >= 0)
    {
      count++; //for subtraction, count--
    }
    print_count();
    cleanup_platform();
    return 0;
}
```

APPENDIX E LOAD-STORE VICTIM CODE

```
int main()
{
    init_platform();
    initialize_variables();
    set_apu_frequency();
    for (int i = 0 ; i < array_size; i++)
    {
      arr1[i] = constant1 - i; //store values in
    arr1
    }
    for (int j = 0; j< array_size; j++)
    {
      arr1[j] = arr1[j] - 2; //load arr1 values
    and store new values
    }
    print_array();
    cleanup_platform();
    return 0;
}
```

APPENDIX F PRINTING VICTIM CODE

```
int main()
{
    init_platform();
    initialize_variables();
    set_apu_frequency();
    for (int i = 0; i < 1000; i++)
    {
      printf("%d \n", i);
    }
    cleanup_platform();
    return 0;
}
```

REFERENCES

[1] J. M. Shalf and R. Leland, "Computing beyond Moore's law," *Computer*, vol. 48, no. 12, pp. 14–23, Dec. 2015.

[2] "Zynq UltraScale+ MPSoC," Xilinx, 2022. [Online]. Available: https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html

[3] "Cyclone V hard processor system technical reference manual," Intel, 2020. [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/cyclone-v/cv_5v4.pdf

[4] "FPGA-based Amazon EC2 F1 computing instances," Amazon AWS, 2022. [Online]. Available: aws.amazon.com/ec2/instance-types/f1/

[5] Alibaba, "Compute optimized instance families with FPGAs," Alibaba, alibabacloud.com/help/doc-detail/108504.htm.

[6] "Machine learning." [Online]. Available: https://azure.microsoft.com/en-us/pricing/details/machine-learning/

[7] S. S. Mirzargar and M. Stojilović, "Physical side-channel attacks and covert communication on FPGAs: A survey," in *29th International Conference on Field-Programmable Logic and Applications*, Barcelona, Spain, Sep. 2019, pp. 202–10.

[8] O. Glamočanin, L. Coulon, F. Regazzoni, and M. Stojilović, "Are cloud FPGAs really vulnerable to power analysis attacks?" in *DATE*, Grenoble, France, Mar. 2020, pp. 1–4.

[9] I. Giechaskiel, K. B. Rasmussen, and J. Szefer, "C3APSULe: Cross-FPGA covert-channel attacks through power supply unit leakage," in *41st Symposium on Security and Privacy (S&P'20)*, San Francisco, CA, USA, May 2020, pp. 1728–41.

[10] M. Zhao and G. E. Suh, "FPGA-based remote power side-channel attacks," in *S&P*, San Francisco, CA, USA, May 2018, pp. 229–44.

[11] D. R. E. Gnad, F. Oboril, and M. B. Tahoori, "Voltage drop-based fault attacks on FPGAs using valid bitstreams," in *27th International Conference on Field-Programmable Logic and Applications*, Ghent, Belgium, Sep 2017, pp. 1–7.

[12] K. Matas, T. M. La, K. D. Pham, and D. Koch, "Power-hammering through glitch amplification – attacks and mitigation," in *28th IEEE Symposium on Field-Programmable Custom Computing Machines*, Fayetteville, AR, USA, May 2020, pp. 65–9.

[13] T. La, K. Pham, J. Powell, and D. Koch, "Denial-of-service on FPGA-based cloud infrastructure - attack and defense," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2021, no. 3, pp. 441–64, Jul. 2021.

[14] D. Mahmoud and M. Stojilović, "Timing violation induced faults in multi-tenant FPGAs," in *DATE*, Florence, Italy, Mar. 2019, pp. 1745–50.

[15] D. G. Mahmoud, W. Hu, and M. Stojilović, "X-attack: Remote activation of satisfiability don't-care hardware Trojans on shared FPGAs," in *30th International Conference on Field-Programmable Logic and Applications*, 2020.

[16] J. Krautter, D. R. E. Gnad, and M. B. Tahoori, "FPGAhammer: Remote voltage fault attacks on shared FPGAs, suitable for DFA on AES," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2018, no. 3, pp. 44–68, Aug. 2018.

[17] P. Qiu, D. Wang, Y. Lyu, and G. Qu, "VoltJockey: Breaking SGX by software-controlled voltage-induced hardware faults," in *AsianHOST*, Xi'an, China, Dec. 2019, pp. 1–6.

[18] ——, "VoltJockey: Breaching TrustZone by software-controlled voltage manipulation over multi-core frequencies," in *ACM SIGSAC Conference on Computer and Communications Security*. London, UK: ACM, 2019, pp. 195–209.

[19] K. Murdock, D. Oswald, F. D. Garcia, J. Van Bulck, D. Gruss, and F. Piessens, "Plundervolt: Software-based fault injection attacks against Intel SGX," in *IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA, USA, May 2020, pp. 1466–82.

[20] Z. Kenjar, T. Frassetto, D. Gens, M. Franz, and A.-R. Sadeghi, "V0LTpwn: Attacking x86 processor integrity from software," in *29th Usenix Security Symposium*, Virtual, Aug. 2020, pp. 1445–61.

[21] M. Lipp, A. Kogler, D. Oswald, M. Schwarz, C. Easdon, C. Canella, and D. Gruss, "PLATYPUS: Software-based power side-channel attacks on x86," in *IEEE Symposium on Security and Privacy (SP)*, Virtual, May 2021, pp. 355–71.

[22] A. Tang, S. Sethumadhavan, and S. Stolfo, "CLKSCREW: Exposing the perils of security-oblivious energy management," in *26th Usenix Security Symposium*, Vancouver, BC, Aug. 2017, pp. 1057–74.

[23] D. G. Mahmoud, S. Hussein, V. Lenders, and M. Stojilović, "FPGA-to-CPU undervolting attacks," in *Proceedings of the 2022 Conference & Exhibition on Design, Automation & Test in Europe*, Virtual Event, Mar. 2022, pp. 999–1004.

[24] L. Zussa, J.-M. Dutertre, J. Clédière, B. Robisson, and A. Tria, "Investigation of timing constraints violation as a fault injection means," in *27th Conference on Design of Circuits and Integrated Systems (DCIS)*, Avignon, France, Nov. 2012, pp. 1–6.

[25] "ARMv8-A power management," ARM, 2022. [Online]. Available: https://developer.arm.com/documentation/100960/0100/

[26] G. Provelengios, D. Holcomb, and R. Tessier, "Power wasting circuits for cloud FPGA attacks," in *30th International Conference on Field-Programmable Logic and Applications*, 2020.

[27] "ZCU102 evaluation board user guide," Xilinx, 2019. [Online]. Available: https://docs.xilinx.com/v/u/en-US/ug1182-zcu102-eval-bd

[28] M. M. Alam, S. Tajik, F. Ganji, M. Tehranipoor, and D. Forte, "RAM-Jam: Remote temperature and voltage fault attack on FPGAs using memory collisions," in *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, Atlanta, GA, USA, Aug. 2019, pp. 48–55.

[29] T. M. La, K. Matas, N. Grunchevski, K. D. Pham, and D. Koch, "FPGADefender: Malicious self-oscillator scanning for Xilinx UltraScale + FPGAs," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 13, no. 3, pp. 15:1–15:31, Sep. 2020.

[30] Y. Luo, C. Gongye, Y. Fei, and X. Xu, "DeepStrike: Remotely-guided fault injection attacks on DNN accelerator in cloud-FPGA," in *58th ACM/IEEE Design Automation Conference (DAC)*. San Francisco, CA, USA: IEEE, Dec. 2021, pp. 295–300.

[31] Z. Weissman, T. Tiemann, D. Moghimi, E. Custodio, T. Eisenbarth, and B. Sunar, "JackHammer: Efficient rowhammer on heterogeneous FPGA-CPU platforms," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 169–95, Jun 2020.

[32] M. Tunstall, D. Mukhopadhyay, and S. Ali, "Differential fault analysis of the advanced encryption standard using a single fault," in *Information Security Theory and Practice. Security and Privacy of Mobile Devices in Wireless Communication*. Heraklion, Greece: Springer, Jun. 2011, pp. 224–33.

[33] P. Dusart, G. Letourneux, and O. Vivolo, "Differential fault analysis on AES," in *Applied Cryptography and Network Security*. Kunming, China: Springer, Oct. 2003, pp. 293–306.

[34] "Genesys ZU: Zynq UltraScale+ MPSoC development board," Digilent, 2022. [Online]. Available: https://digilent.com/reference/programmable-logic/genesys-zu/reference-manual

[35] "ZCU104 evaluation board user guide (UG1267)," 2018. [Online]. Available: https://docs.xilinx.com/v/u/en-US/ug1267-zcu104-eval-bd

[36] Z. Chen, G. Vasilakis, K. Murdock, E. Dean, D. Oswald, and F. D. Garcia, "VoltPillager: Hardware-based fault injection attacks against Intel SGX enclaves using the SVID voltage scaling interface," in *30th Usenix Security Symposium*, Vancouver, Canada, Aug. 2021, pp. 1–18.

[37] "MAX15303 PMBus command set user's guide," Maxim Integrated. [Online]. Available: https://pdfserv.maximintegrated.com/en/an/UG5816.pdf

[38] "Zynq UltraScale+ device technical reference manual," 2020. [Online]. Available: https://docs.xilinx.com/v/u/en-US/ug1085-zynq-ultrascale-trm

[39] J. Gravellier, J.-M. Dutertre, Y. Teglia, P. L. Moundi, and F. Olivier, "Remote side-channel attacks on heterogeneous SoC," in *Smart Card Research and Advanced Applications*, Prague, Czech Republic, Nov. 2019, pp. 109–125.

[40] K. M. Zick, M. Srivastav, W. Zhang, and M. French, "Sensing nanosecond-scale voltage attacks and natural transients in FPGAs," in *21st ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, Monterey, CA, USA, 2013, pp. 101–04.

[41] S. Moini, S. Tian, D. Holcomb, J. Szefer, and R. Tessier, "Remote power side-channel attacks on BNN accelerators in FPGAs," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Grenoble, France, Feb. 2021, pp. 1639–44.

[42] S. Moini, A. Deric, X. Li, G. Provelengios, W. Burleson, R. Tessier, and D. Holcomb, "Voltage sensor implementations for remote power attacks on FPGAs," *ACM Trans. Reconfigurable Technol. Syst.*, aug 2022, just Accepted. [Online]. Available: https://doi.org/10.1145/3555048

[43] A. Kogler, D. Gruss, and M. Schwarz, "Minefield: A software-only protection for SGX enclaves against DVFS attacks," in *31st USENIX Security Symposium*, Boston, MA, Aug. 2022, pp. 4147–64.

[44] "UltraScale architecture system monitor user guide," Sep. 2021.

[45] "ARM cortex-A series programmer's guide for ARMv8-A," Mar. 2015.

[46] J. Gravellier, J.-M. Dutertre, Y. Teglia, and P. L. Moundi, "FaultLine: Software-based fault injection on memory transfers," in *2021 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, Tysons Corner, VA, USA, Dec. 2021, pp. 46–55.

[47] B. Selmke, F. Hauschild, and J. Obermaier, "Peak clock: Fault injection into PLL-based systems via clock manipulation," in *Proceedings of the 3rd ACM Workshop on Attacks and Solutions in Hardware Security*, London, United Kingdom, Nov. 2019, pp. 85–94.

[48] G. Piret and J.-J. Quisquater, "A differential fault attack technique against SPN structures, with application to the AES and KHAZAD," in *Cryptographic Hardware and Embedded Systems - CHES 2003*, vol. 2779, Cologne, Germany, Sep. 2003, pp. 77–88.

[49] K. Murdock, D. Oswald, F. D. Garcia, J. V. Bulck, F. Piessens, and D. Gruss, "Plundervolt: How a little bit of undervolting can create a lot of trouble," *IEEE Security & Privacy*, vol. 18, no. 5, pp. 28–37, Sep. 2020.

[50] A. Barenghi, L. Breveglieri, I. Koren, and D. Naccache, "Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures," *Proceedings of the IEEE*, vol. 100, no. 11, pp. 3056–76, Nov. 2012.

[51] J. Krautter, D. R. E. Gnad, and M. B. Tahoori, "Mitigating electrical-level attacks towards secure multi-tenant FPGAs in the cloud," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 12, no. 3, pp. 1–26, Sep. 2019.

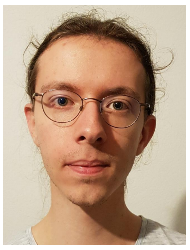
[52] ——, "Remote and stealthy fault attacks on virtualized FPGAs," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, Feb. 2021, pp. 1632–7.

[53] G. Provelengios, D. Holcomb, and R. Tessier, "Characterizing power distribution attacks in multi-user FPGA environments," in *29th International Conference on Field-Programmable Logic and Applications*, Barcelona, Spain, Sep. 2019, pp. 194–201.

[54] S. S. Mirzargar, G. Renault, A. Guerrieri, and M. Stojilović, "Nonintrusive and adaptive monitoring for locating voltage attacks in virtualized FPGAs," in *2020 International Conference on Field-Programmable Technology (ICFPT)*, Maui, HI, USA, Dec. 2020, pp. 288–9.

[55] H. Nassar, H. AlZughbi, D. R. E. Gnad, L. Bauer, M. B. Tahoori, and J. Henkel, "LoopBreaker: Disabling interconnects to mitigate voltage-based attacks in multi-tenant FPGAs," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, Munich, Germany, Nov. 2021, pp. 1–9.

DINA G. MAHMOUD (Student Member, IEEE) received the B.Sc. degree in electronics and communications engineering from the American University in Cairo, Egypt in 2019. She is currently pursuing her Ph.D. in computer and communication sciences at EPFL, Lausanne, Switzerland. She is the first recipient of the Cyber-Defence (CYD) Campus Doctoral Fellowship and the recipient of the Google Generation Scholarship. Her research interests include the hardware security of FPGA-CPU heterogeneous systems.



DAVID DERVISHI received a B.Sc. degree in computer science from EPFL, Lausanne, Switzerland, in 2021. He is currently pursuing a joint M.Sc. degree in cybersecurity at EPFL and ETH Zürich.

SAMAH HUSSEIN received her B.Sc. degree in computer engineering from the American University in Cairo in 2022. She worked as a research intern at the EPFL School of Computer and Communication Sciences for three months, and she is currently pursuing her PhD there. Her current research interests include hardware security and reconfigurable computing.

VINCENT LENDERS received his M.Sc. and Ph.D. degrees in electrical engineering and information technology from ETH Zurich, in 2001 and 2006, respectively. After his Ph.D., he was postdoctoral researcher at Princeton University. In 2008, he joined armasuisse where he is currently the Director of the Cyber-Defence Campus. His research interests lay at the intersection between cyber security, data science, networking, and crowdsourcing. Over the past 15 years, he has published over 150 scientific publications in these areas and has contributed to the development of various cyber security and information systems which have been adopted by the Swiss Federal Department of Defence. He is also the co-founder and member of the board of the OpenSky Network and Electrosense Associations.

MIRJANA STOJILOVIĆ (Senior Member, IEEE) received the Dipl. Ing. and Ph.D. degrees from the School of Electrical Engineering, University of Belgrade, in 2006 and 2013, respectively. She joined the School of Computer and Communication Sciences at EPFL in 2016, where she leads a research group working on electronic design automation, reconfigurable computing, and hardware security. She serves on the program committees of FPGA, FPL, FCCM, and DATE conference. She is associate editor for IEEE Embedded System Letters and in the process of joining the editorial board of ACM Transactions on Reconfigurable Technology and Systems. In 2021, she was on the Best Paper Award (BPA) committee of the FPGA conference. In 2020, she was nominated for the BPA at the International Conference on Field-Programmable Technology (FPT). Additionally, she received the BPA at EMC Europe 2016, Young Scientist Award at ICLP'16, and the Young Author BPA at TELFOR'12. In 2015, the EPFL School of Computer and Communication Sciences presented her with the Teaching Award. Mirjana is principal investigator in the Swiss National Foundation (SNF) funded project *Secure FPGAs in the Cloud*.

• • •