IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 14 December 2023, accepted 4 January 2024, date of publication 11 January 2024, date of current version 22 January 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3353134

RESEARCH ARTICLE

X-Attack 2.0: The Risk of Power Wasters and Satisfiability Don't-Care Hardware Trojans to Shared Cloud FPGAs

DINA G. MAHMOUD[1], (Member, IEEE), BEATRICE SHOKRY[1], VINCENT LENDERS[2], (Member, IEEE), WEI HU[3], (Member, IEEE), AND MIRJANA STOJILOVIĆ[1], (Senior Member, IEEE)

[1]School of Computer and Communication Sciences, EPFL, 1015 Lausanne, Switzerland
[2]Cyber-Defence Campus, armasuisse, 3602 Thun, Switzerland
[3]School of Cybersecurity, Northwestern Polytechnical University, Xi'an 710072, China

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

This work was supported by armasuisse Science and Technology.

ABSTRACT Cloud computing environments increasingly provision field-programmable gate arrays (FPGAs) for their programmability and hardware-level parallelism. While FPGAs are typically used by one tenant at a time, multitenant schemes supporting spatial sharing of cloud FPGA resources have been proposed in the literature. However, the spatial multitenancy of FPGAs opens up new attack surfaces. Investigating potential security threats to multitenant FPGAs is thus essential for better understanding and eventually mitigating the security risks. This work makes a notable step forward by systematically analyzing the combined threat of FPGA power wasters and satisfiability don't-care hardware Trojans in shared cloud FPGAs. We demonstrate a successful remote undervolting attack that activates a hardware Trojan concealed within a victim FPGA design and exploits the payload. The attack is carried out entirely remotely, assuming two spatially colocated FPGA users isolated from one another. The victim user's circuit is infected with a Trojan, triggered by a pair of *don't-care* signals that never reach the combined trigger condition during regular operation. The adversary, targeting the exploitation of the Trojan, deploys power waster circuits to lower the supply voltage of the FPGA. The assumption is that, under the effect of the lowered voltage, don't-care signals may reach the particular state that triggers the Trojan. We name this exploit X-Attack and demonstrate its feasibility on an embedded FPGA and real-world cloud FPGA instances. Additionally, we study the effects of various attack tuning parameters on the exploit's success. Finally, we discuss potential countermeasures against this security threat and present a lightweight self-calibrating countermeasure. To the best of our knowledge, this is the first work on undervolting-based fault-injection attacks in multitenant FPGAs to demonstrate the attack on commercially available cloud FPGA instances.

INDEX TERMS FPGA security, hardware trojans, multitenancy, timing faults, remote attack.

I. INTRODUCTION

Individuals and companies increasingly adopt cloud computing for a variety of use cases. Instead of acquiring and managing their hardware and software, users can rent computing instances from cloud service providers (CSPs) and use them to test and deploy their applications [1]. Given the widespread use of the cloud, the security of cloud-based systems is now of the utmost importance. In particular, the systems provided by the CSPs deal with user data and must guarantee its integrity and confidentiality. With the increasing variety and heterogeneity of cloud computing instances, security requirements are no longer limited to storage and central processing units (CPUs) but also include specialized hardware, such as graphics processing units (GPUs) and field-programmable gate arrays (FPGAs).

FPGAs have emerged as hardware platforms well-suited for both regular and irregular forms of application

The associate editor coordinating the review of this manuscript and approving it for publication was Christian Pilato.

parallelism. Unlike other computing units, they support fine-grained (bit-level) hardware design specification and reconfiguration at runtime. FPGAs are powerful hardware accelerators for machine learning, big data, and cryptographic applications [2], [3], [4]. Cloud service providers such as Amazon, Alibaba, and Baidu offer clients the possibility to rent the FPGA instances to design and deploy their accelerators [5], [6], [7], [8]. Microsoft uses FPGAs in their datacenters to accelerate some of their services (e.g., Microsoft Bing) [9], [10].

For better and more efficient cloud utilization, remote users can share the computing resources. Sharing is usually achieved by means of virtualization, providing the users with virtual instances that behave similarly to physical ones. Given that multiple virtual instances can use the same hardware, the CSPs can manage the resource allocation each user needs. Another advantage of virtualization is that it is cost-effective and helps with recovering from failures [11]. The benefits of virtualization are the reason for the increasing interest in developing solutions for cloud FPGA virtualization and sharing [12], which often aim to support both temporal and spatial multitenancy.

The prospect of multitenancy has pushed researchers and the industry to start investigating related security risks that could affect cloud users. Security exploits involving CPUs typically target shared memory, microarchitecture, and the interconnect [13], [14], [15]. FPGAs are not immune to the risks related to hardware sharing, either. Researchers have shown attacks exploiting the underlying FPGA power distribution network (PDN) and the communication interfaces [16], [17] to gain side-channel information about the colocated FPGA tenant. Some of the attacks are less subtle: they use the shared PDN as a medium for propagating disturbances (e.g., glitches) with the intent of causing a denial of service (DoS) or injecting faults in the victim's operation; sometimes, faults may lead to the recovery of secret encryption keys or degradation of the accuracy of neural network inference [18], [19].

Issues arising from multitenancy aside, FPGAs are also vulnerable to supply chain attacks. The design and manufacture of FPGAs suffer from the same trust and security concerns known to other integrated circuits [20]. Even if no tampering occurs during the chip manufacturing, cloud users may end up with erroneous or malicious FPGA applications due to the use of untrusted third-party intellectual property (IP) cores or insufficiently verified design modules. If any of the parties supplying the modules is untrusted or compromised, the insertion of malicious functionality in the form of hardware Trojans can compromise security. Additionally, if the designs run within a cloud instance and the CSP is untrusted or compromised, security concerns may arise at that stage as well [21].

While the literature examines the possibility of stealthy hardware Trojan insertion or undervolting for fault injection separately, we note that the two threats combined lead to another equally relevant attack model. In our earlier work, we have shown for the first time that an FPGA-based adversary can activate a stealthy satisfiability don't-care (SDC) hardware Trojan by generating power supply voltage disturbances [22]. We named this attack X-Attack after the don't-care conditions that act as the Trojan trigger.

Satisfiability don't cares are combinations of internal circuit signals that, by construction, can never occur during regular operation (not counting transient glitches that do not affect the computation results). An SDC Trojan is, therefore, a hardware Trojan that uses SDC signals as a trigger. In our previous work [22], we first showed that FPGA power supply voltage manipulations can make SDC signals take unexpected values, potentially triggering the SDC Trojan. Then, we demonstrated a successful key-recovery attack against an advanced encryption standard (AES) core, previously compromised by inserting an SDC Trojan.

Within the context of FPGA-based undervolting attacks, X-Attack adds to the existing body of knowledge. FPGA-based undervolting was first used for denial-of-service (DoS) exploits [23]. Later, researchers found ways to better control the voltage variations to avoid DoS and, instead, perform fault injection. Successful examples of fault-injection attacks include differential fault analysis (DFA) attacks against AES or intentional neural network performance degradation [18], [19]. X-Attack, however, is an example of a remote FPGA attack targeting a hardware Trojan; unlike a DFA attack, X-Attack can recover an arbitrary secret payload that the SDC Trojan is configured to leak, and X-Attack does not require the adversary to have access to the victim inputs (in comparison, DFA requires the adversary to be able to send plaintexts). Within the context of hardware Trojan exploits, previous work focused on activating SDC Trojans through physical access to the device [24]. Instead, X-Attack shows that the SDC Trojan can be triggered remotely without physical access to the device.

Our earlier paper presented the X-Attack proof-of-concept on an FPGA development board in a laboratory environment [22]. No experiments were performed on a real cloud FPGA instance. Opting for an FPGA development board is a frequent practice in this line of research because it is less costly and because the adversary has complete control of the hardware and environmental conditions, plus the freedom to cause an FPGA reset unintentionally without consequences. To our knowledge, no prior work on remote FPGA fault injection confirmed their findings on an actual commercial cloud FPGA instance. Today, the security community strongly recommends validating the vulnerabilities on the cloud, where the PDN is more robust [25]. In comparison, DoS and side-channel attacks have been experimentally validated on the Amazon AWS cloud instances equipped with AMD FPGAs [26], [27]. In this manuscript, X-Attack 2.0, we extend our previous publication with experiments targeting an actual cloud instance. We demonstrate a successful exploit,

confirming the relevance of X-Attack.[1] Additionally, instead of AMD FPGAs, we target Intel FPGAs. Our analyses provide novel and detailed insights into undervolting effects on a cloud instance with Intel FPGAs,[2] thus complementing the existing literature focusing on AMD FPGAs [28].

Specifically, this manuscript extends our previous work [22] as follows:

- It demonstrates X-Attack, a novel exploit combining remote undervolting with SDC Trojans, in an actual cloud setting involving an Intel FPGA. The attack target is an AES core with an integrated SDC hardware Trojan. The attack goal is to steal the secret AES key, by triggering the Trojan and recovering the payload. To the best of our knowledge, this is the first work to show successfully injected and exploited FPGA-undervolting-based faults on commercially deployed cloud FPGA instances.
- We systematically explore and evaluate the attack parameters and their effect on the attack success, focusing on the attacker size, the way it is activated and controlled, and the distance between the attacker and the victim. The goal is to learn how to adapt the exploit to various targets.
- We analyze the relationship between various SDC conditions (Trojan triggers), the victim AES circuit operation, and the success of X-Attack.
- We implement, test, and validate a self-calibrating version of a lightweight countermeasure [22] against X-Attack and similar fault-injection exploits.

The remainder of this paper is organized as follows. First, Section II presents the necessary background. The threat model is discussed in Section III. Section IV focuses on X-Attack. The experimental setup and the results are presented in Sections V and VI, respectively. Section VII addresses potential countermeasures, whereas Section VIII discusses the obtained results. Section IX presents the related work. Finally, Section X concludes the paper.

II. BACKGROUND

In this section, we present the necessary background for X-Attack on embedded and cloud FPGAs. We begin with an overview of FPGA cloud offerings and how they differ from those used in embedded settings. We then present SDC Trojans. Then, we discuss AES, the victim we use for the proof-of-concept for X-Attack. Finally, we present remote undervolting-based exploits on FPGAs.

A. CLOUD VS. EMBEDDED FPGAS

Thanks to their integration in cloud computing instances and use for acceleration, FPGAs evolved from prototyping platforms to widely available acceleration platforms. The adoption of FPGAs by CSPs required developing the necessary infrastructure to ensure efficient management of the FPGA resources and their incorporation into the cloud ecosystem.

In datacenters and the cloud, FPGAs are typically incorporated as accelerator cards connected through a peripheral component interconnect express (PCIe) bus to the CPU instances [29]. On the other hand, embedded offerings typically combine the FPGA with a CPU in the same package, resulting in a system-on-chip (SoC) or multi-processor SoC (MPSoC). In an SoC, the FPGA and the CPU can communicate through the advanced extensible interface (AXI) [30]. In the cloud, FPGA logic resources are typically split into the *shell* and the *user* partitions. The shell, controlled by the CSP, manages input/output (I/O) interfaces and the data exchange between the user design and the rest of the system. Through the shell management, the CSP controls resource allocation and user isolation [7]. The division of the logic into shell and user regions also means that in cloud settings, differently from embedded settings, user applications have less control over the underlying hardware and interfaces.

Major CSPs commonly deploy custom-made FPGA boards [7]. In the absence of publicly available implementation details of the cloud FPGA instances, an adversary aiming to exploit the PDN has no alternative but to experiment with an off-the-shelf FPGA card. However, the attack tuned for one board is likely challenging to port to another because the underlying PDNs differ. Accordingly, preparing an exploit against a custom-made cloud FPGA is challenging and requires online testing. Besides the superior PDN, cloud FPGAs are often built in the latest technology nodes and offer significantly more resources than smaller embedded FPGAs (the improved technology and larger size are other reasons why the multitenancy model is a desirable direction for cloud FPGAs). All the differences between cloud and embedded FPGAs pose a challenge in porting exploits between the two deployment scenarios, which can only be addressed by running experiments in both settings.

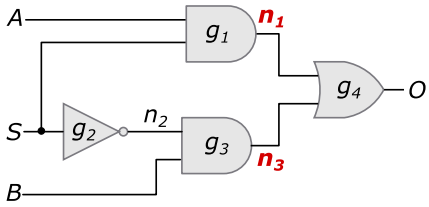
B. SATISFIABILITY DON'T-CARE HARDWARE TROJANS

Hardware Trojans are malicious hardware modifications aiming to disable a system or leak its secrets when triggered [31]. Adversaries can implant Trojans in application-specific integrated circuits (ASICs), microcontrollers, third-party IPs, and by modifying FPGA bitstreams. By design, the malicious party implements stealthy and rarely-activated Trojans to avoid their detection [31]. Our work considers Trojans that can break a circuit's confidentiality by leaking its secrets to an adversary. Specifically, we focus on satisfiability don't-care Trojans [24] described below.

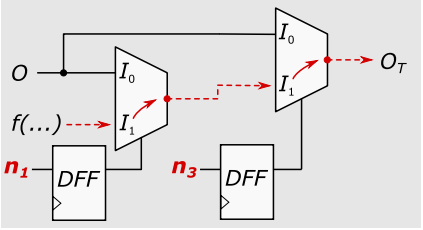
Due to the way digital hardware circuits are designed and built, there exist correlations among the values of internal signals. Specifically, because of the shared inputs that affect the values of many internal signals, the logic functions

[1]For clarity and because the underlying attack mechanism is shared between this and previous work, we refer to the exploit as X-Attack throughout the manuscript.

[2]We do not disclose the name of the CSP because the possibility to inject faults in other FPGAs and other CSPs cannot be entirely excluded. To our knowledge, the target CSP currently does not support spatial FPGA sharing.



(a) Multiplexer



(b) SDC Trojan

FIGURE 1. Example of SDC condition and its use in the SDC Trojan. (a) Gate-level implementation of a 2-to-1 multiplexer with the SDC condition that n_1 and n_3 cannot be 11 (not accounting for transient glitches). (b) Implementation of SDC Trojan using the trigger signals n_1 and n_3.

produce signals that are correlated. These correlations may lead to signals never reaching some internal states. Such unreachable states are known as satisfiability don't-cares: states that will never occur under any input sequence during normal operation.

For example, let us consider the gate-level implementation of a 2-to-1 multiplexer (MUX), as shown in Fig. 1. The boolean function of the MUX is $\mathtt{O} = \mathtt{S}\cdot\mathtt{A} + \overline{\mathtt{S}}\cdot\mathtt{B}$, where A and B are the inputs, S is the select line, and O is the multiplexer output. Since they share the common select line S, the two AND gates (g_1 and g_3) produce outputs (n_1 and n_3) that are correlated. In this case, the correlation is such that the outputs can never simultaneously be logical 1. Consequently, we can say that the signal pair (n_1, n_3) represents an SDC condition such that the equality $(n_1, n_3) = (1, 1)$ will never be observed under normal operating conditions (not considering transient states when the logic is switching).

In the simple example of a MUX, the SDC condition is related to signals within the same combinational block and where the condition's origin is two gates sharing the same signal. When the SDC condition is localized in this way, it is termed a *local* SDC, and satisfiability analysis can easily identify it. However, within a large design, far-apart signals can still be correlated at a *global* level. For instance, when one of the circuit's primary inputs is a logical 1, the path correlations will lead to some internal signals never being 0. The concept of an SDC differs from an *external* don't-care condition, which arises from incomplete design specifications, and is accordingly easy to eliminate. On the other hand, SDCs are widely spread within completely specified hardware functions and cannot be eliminated [24].

Seeing how widespread SDCs are, Hu et al. introduced the idea of using an SDC signal pair as trigger signals for a stealthy hardware Trojan [24]. Following the example of the MUX in Fig. 1a, we can build the Trojan in Fig. 1b and use n_1 and n_3 as its triggers. The way the Trojan is built, the output of the circuit with the Trojan (O_T) is always equivalent to the output of the circuit without it (O) as long as the operating conditions are normal. This equivalence makes the Trojan difficult to detect. Furthermore, each trigger signal can switch freely, so a switching probability analysis will not point to an issue with either trigger signal. Finally, using flip-flops (FFs) for the Trojan trigger signals guarantees that no transient glitches will accidentally activate the Trojan. The FFs also separate the combinational block where the signals originate from that in which the Trojan is implemented. This separation does not change the functionality but helps prevent the synthesis tools from optimizing away the Trojan.

The design proposed by Hu et al. relies on injecting a fault into the operation of the circuit to have the trigger signals reach the SDC condition [24], which is the only way to activate the Trojan and have it leak a secret (payload) to the output. To activate the Trojan, Hu et al. externally manipulated the clock to inject glitches [24]. However, as physical access to the FPGA device and control over the victim's clock are not always possible, regardless of whether the target is an embedded or cloud FPGA, we choose remote undervolting as the fault injection approach [22]. To perform remote FPGA undervolting, an adversary requires FPGA power wasters, which will be explained in Section II-D.

C. AES

Symmetric encryption, which uses one key for encrypting and decrypting the data, is a fast and efficient way to secure data. The AES algorithm is among the most widely used symmetric encryption algorithms. Modern devices implement AES as a crypto-accelerator or a software crypto-library [32].

AES is a block cipher: it operates on 128-bit blocks of data. The key size varies depending on the security level; typically, the key has 128, 192, or 256 bits. The AES algorithm, illustrated in Fig. 2, takes the input plaintext (the 128 bits to be encrypted) and applies round transformations multiple times to obtain the output ciphertext (scrambled data that cannot be read without the key). The exact number of rounds depends on the chosen key size. For a 128-bit key, the encryption takes ten rounds. The rounds consist of the following operations: AddRoundKey, MixColumns (not performed in the last round), SubBytes, and ShiftRows [33].

Due to the widespread use of AES, many hardware AES implementations are available, and designs requiring AES can leverage third-party IPs. When implementing AES in hardware, the designer may opt for a lightweight circuit by building one module for a round and executing each round sequentially using the same module. To increase the throughput, the designer can instantiate a module for each round and pipeline the circuit, similarly to Fig. 2. FPGA implementations of AES often leverage reconfigurable resources for circuit optimization. For example, FPGA memory elements and look-up tables (LUTs) can effectively

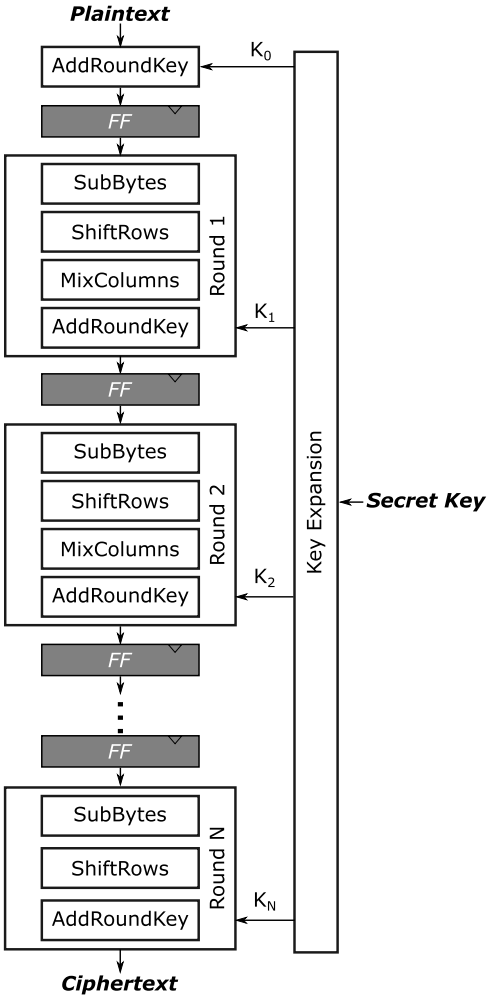


FIGURE 2. Illustration of the AES algorithm with the rounds pipelined.

implement a substitution box (Sbox) for the SubBytes operation.

Another consequence of AES's popularity is that AES hardware and software implementations are among the most common attack targets. Adversaries can use side channels or fault injection to recover the AES secret key. When using fault injection, attackers typically aim to introduce a fault that affects the algorithm output. Then, for instance, they can perform a differential fault analysis (DFA) by comparing the faulty and the correct ciphertext to reduce the search space of the secret key [33]. Our work targets fault injection, but instead of DFA, it uses an SDC hardware Trojan hidden in the AES core to recover the secret encryption key.

D. REMOTE UNDERVOLTING-BASED FAULT INJECTION

With the increasing use of cloud computing and the Internet of Things (IoT), remote device access has become a relatively common feature. Consequently, researchers have been studying the associated security risks, particularly the exploits that leverage software access (including FPGA configuration) for fault injection. Specifically, researchers

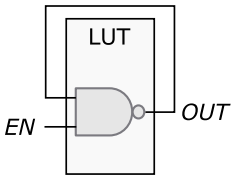


FIGURE 3. A single-LUT RO with an enable signal.

have examined the possibility of using the programmable logic of an FPGA to build malicious circuits known as *power wasters*; when active, these circuits draw excessive current and cause on-chip voltage drops. As a consequence of reduced voltage, the circuit delays increase. If the increase is significant enough, timing constraints may be violated, resulting in a faulty operation [34].

Various implementations of FPGA power wasters have been presented in literature [35]. Given that dynamic power consumption increases with the switching frequency, fast oscillators such as combinational ring oscillators (ROs) emerged as some of the most effective power wasters [36]. An RO is built by chaining an odd number of inverters and connecting the output of the last one to the input of the first, thus creating a closed loop. On an FPGA, an RO can be built using a single LUT as shown in Fig. 3 [23]. Here, the LUT implements the NAND functionality. Its output is connected back to one of its inputs, creating an oscillator. The second input is used to enable and disable the oscillatory behavior. The lower the number of inverters in an RO, the shorter the combinational path and the higher the oscillation frequency. The effectiveness of this single-stage RO can be further improved by increasing the output capacitive load (e.g., by connecting the RO output to free LUT inputs or FPGA routing wires) [26], [37]. Such power wasters use the available hardware resources better: they complement logical with routing resources. However, as they consume higher power, they are more likely to trigger a permanent fault (e.g., reset), resulting in a DoS attack. The more effective the power wasters, the finer the control level required.

By analyzing the design netlists, combinational loops can be detected. Detection and removal of combinational loops can be incorporated into the FPGA compilation flow; however, netlist screening is not a standard practice yet. An exception is Amazon AWS [5], where FPGA combinational loops are detected before generating the bitstreams. The issues with preventing combinational loops by default are twofold. First, benign circuits using combinational loops (e.g., physical unclonable functions (PUFs) or RO-based true random number generators) would not be supported. Second, the absence of a combinational loop is far from a guarantee against fault-injection attacks. Breaking the combinational loop with a flip-flop (FF) or latch makes the resulting RO pass the screening [38]. Other examples of effective and combinational-loop-free power wasters include glitch generators and glitch amplifiers, block random access memories (BRAMs), and overclocked block ciphers [26], [38], [39], [40].

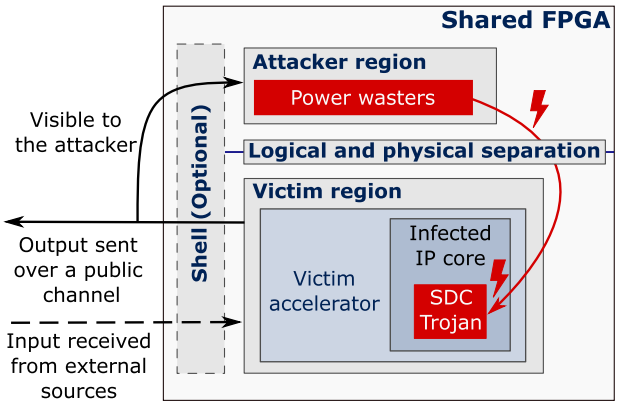


FIGURE 4. Threat model, with optional components represented by dashed edges.

Instantiating and simply enabling the power wasters is insufficient for a successful attack. An adversary needs to carefully control the activity of the power wasters to avoid consuming more power than the power supply can provide (i.e., avoid FPGA reset [23]). Such control requires extensive experimentation, analysis of the obtained effects, and tuning of the enable signal pattern. Carefully controlled fault injection was shown to be effective for biasing true random number generators, and DFA exploits against AES running in hardware or software [18], [41], [42].

III. THREAT MODEL

Fig. 4 illustrates the threat model we consider in this work. The two main aspects are FPGA multitenancy and the existence of a hardware Trojan. Following other works focusing on hardware Trojans, our threat model assumes that the target victim uses a hardware module (e.g., an IP core) previously compromised by a hardware Trojan [22], [24], [43]. For example, the victim uses a black-box module offered by the cloud service provider, the FPGA manufacturer, or an untrusted third party. We assume that a malicious party has interfered with the production of the IP and has hidden a stealthy Trojan within. The victim trusts the IP and does not have access to a golden-reference IP without the Trojan; thus, they cannot compare the netlist to detect the existence of the Trojan. The design with the SDC Trojan is functionally equivalent to a Trojan-free design, and therefore, functional checks do not detect the existence of the Trojan [24].

The exploit under consideration occurs in a multitenant scenario, possibly in a cloud instance. The FPGA is remotely accessible to the attacker and the victim. Each has a private partition where arbitrary designs can be loaded. The partitions are physically and logically separated. The cloud shell can control all shared resources other than the PDN, as in current cloud instances. The victim runs a security-sensitive service, such as encryption, and leverages an IP where an SDC Trojan [24] has been inserted during the design phase. The victim does not implement specially tailored countermeasures against fault injection attacks; it relies on FPGA voltage supervisors and the cloud service provider. The attacker can be the same party that inserted the Trojan or another adversary (collaborative attack) who gained knowledge of the existence of the Trojan. Through the shared PDN, the adversary aims to influence the chip's voltage to trigger the Trojan.

Similarly to differential fault analysis and Trojan-based exploits [18], [24], [42], [44], an additional requirement is for the adversary to be able to observe the Trojan payload: in our case, the output of the compromised IP. For example, the victim can have a public interface for receiving requests from other parties, including potentially malicious ones [18], [24]. The adversary, sending requests to the victim, can control the input and receive back the output. If the victim's IP core is an encryption module, the encrypted data can be sent over an unprotected or shared communication interface, which the adversary may be able to observe. The encrypted data may also be stored in a region of memory where other users can read it, as the encryption should normally provide enough protection to guarantee that the plaintext cannot be recovered from the ciphertext [45]. Finally, the attacker may leverage side channels (e.g., through the host CPU or shared memory) to infer the output [15].

IV. X-ATTACK DESIGN

This section presents X-Attack, our exploit combining undervolting with SDC Trojans. The attack has two main components: the stealthy SDC Trojan hidden within the victim IP and the power-wasting circuits used to attack the IP in a multitenant scenario. We begin by explaining the Trojan design and how an adversary would insert it into the infected IP to avoid detection and increase the likelihood of the attack's success. Then, we discuss the properties of the target victim. Finally, we discuss the attacker design and implementation.

A. TROJAN DESIGN

As in our previous work [22], we use the SDC Trojan proposed by Hu et al. [24]. The Trojan is to be inserted as illustrated in Fig. 5. With access to the victim design's netlist, the adversary, aiming to hide the Trojan, can use a satisfiability (SAT) solver to find signals that together form an SDC signal pair to be used as the Trojan trigger (dc_0 and dc_1 in Fig. 5). We use the SAT solver in Yosys [46]. Without loss of generality, we choose an AES IP core as the victim (more details will follow shortly, in Section V). The search for the signal pair can be repeated multiple times with various criteria to find a suitable pair of signals. One requirement is that the two triggers can take on all combinations other than the SDC condition. Another is that the SDC signals should not be on the critical path to avoid easy detection of the Trojan, if delay-based hardware Trojan detection methods were to be used on the circuit [24]. However, their delays should not be too short either; otherwise, the SDC signals may not be affected by

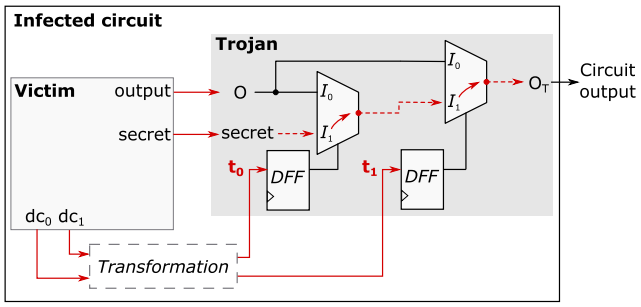


FIGURE 5. Illustration of a circuit infected with an SDC Trojan, showing the connections between the original victim function and the Trojan and the optional transformations impacting don't-care signal delays and polarity.

the attack. If the SDC signal delays are on the short side, the adversary can, within certain limits, modify the victim netlist to increase them (the *Transformation* in Fig. 5). The netlist modification must respect the SDC signals' ability to switch freely and ensure the SDCs are not on the critical path.

When searching for the trigger signal pair, a valid SDC condition is any combination of the two bits (i.e., `00`, `01`, `10`, or `11`). Additionally, whatever SDC condition is found, it can be used as is or inverted. We will later use the inversion to test if the change in the direction of the fault (e.g., the transition from `0` to `1` instead of from `1` to `0`) and the slight change in the SDC delays contribute to the attack success. Consequently, there are eight SDC conditions: `00`, 00_{inv}, `01`, 01_{inv}, `10`, 10_{inv}, `11`, and 11_{inv}. The condition 00_{inv} is obtained by inverting the polarity of the signals forming the SDC condition `11` and inverting the polarity of the multiplexers accordingly. The Trojan with the SDC condition `10` is the mirror image of the `01` Trojan, as reversing the order of the trigger signals will reverse the SDC condition. Therefore, it suffices to focus on `00`, 00_{inv}, `01`, 10_{inv}, `11`, and 11_{inv}. We define two Trojan variants [22]:

- *Trojan-0*: without transformations (as in Fig. 1b).
- *Trojan-delay*: Trojan-0 with extra delay on the trigger path (added by the *Transformation* block in Fig. 5).

The former Trojan variation uses the trigger signals (or their inverse) as found by the SAT solver. The latter increases the signal pair's delays. The increase can be achieved by adding buffers before the registers in Fig. 5. However, for a stealthier design, the original SDC signals can be transformed by combining them with other internal signals. The exact transformation depends on the original SDC condition, as the transformation needs to maintain the same SDC condition and maintain the ability of the two trigger signals to switch freely. Finally, the adversary can also opt for combining transformations with buffers. We derive the following logical expressions for increasing the SDC signal pair delays:

$$\begin{aligned} t_0^{00} &= (dc_0^{00} + dc_1^{00}) \cdot (x + y), \\ t_1^{00} &= (dc_0^{00} + dc_1^{00}) \cdot (\overline{x} + y). \end{aligned} \tag{1}$$

$$\begin{aligned} t_0^{01} &= (dc_0^{01} \cdot \overline{dc_1^{01}}) + (x \cdot y), \\ t_1^{01} &= (\overline{dc_0^{01}} + dc_1^{01}) \cdot (x + y). \end{aligned} \tag{2}$$

$$\begin{aligned} t_0^{10} &= (dc_0^{10} + \overline{dc_1^{10}}) \cdot (x + y), \\ t_1^{10} &= (\overline{dc_0^{10}} \cdot dc_1^{10}) + (x \cdot y). \end{aligned} \tag{3}$$

$$\begin{aligned} t_0^{11} &= (dc_0^{11} \cdot dc_1^{11}) + x \cdot y, \\ t_1^{11} &= (dc_0^{11} \cdot dc_1^{11}) + \overline{x} \cdot y. \end{aligned} \tag{4}$$

Here, t_0 and t_1 are the new trigger signals, while dc_0 and dc_1 are the original don't-care signals. The superscript indicates the SDC condition; for example, the expression in (1) shows the transformation when the SDC condition is `00`. x and y are arbitrary victim signals chosen to increase the delays of the transformed don't-care signals. The transformations use the knowledge of the values that the don't-care signals cannot take and ensure that the resulting signals (post-transformation) cannot attain them either, while freely taking on the other possible values. Since the transformations produce signals that preserve the don't-care condition, they can also be applied recursively.

Once the trigger signals are chosen, the adversary routes them to the two FFs, as shown in Fig. 5. Consequently, faults injected through undervolting will render the outputs of the FFs faulty. They, in turn, are the select signals for the MUXes controlling the output of the victim IP. Depending on the SDC condition, the inputs to the MUXes are arranged such that for the three combinations of the don't-care signals that are not the SDC condition, the output will match the output of the IP without the Trojan. However, if the two select lines reach the SDC condition, the output will be the secret the adversary wishes to leak. In our case, that secret is the encryption key of an AES module.

B. ATTACKER DESIGN

In the multitenant FPGA setup, the adversary aims to inject faults into the IP's operation to reach the SDC condition and trigger the Trojan. Given the absence of physical access to a cloud FPGA, the attacker cannot manipulate the victim's clock signal. Instead, they can manipulate the on-chip voltage; hence, the exploit leverages power-wasting circuits.

First, the adversary decides on the power waster to use (e.g., ROs, register-based ROs, encryption rounds) [26], [38]. If the remotely accessible FPGA has no limitations on the circuits that can be instantiated (e.g, not forbidding combinational loops or not restricting access to phase-locked loops (PLLs)), the malicious party can design power wasters freely. The choice can be made according to available resources, the quality of the PDN (and, accordingly, the required power consumption), and the attacker's experience.

We instantiate the power wasters in N_B *blocks* of N_W power wasters (the exact values will be given in Section V). The power wasters within each block share the enable signal, and the control for each enable signal is independent of other enable signals. We control the start and duration of the attack, whether the signal is continuously active or toggling, and, if toggling, the signal's period and duty cycle. The period determines the number of clock cycles of each repetition

of the toggling of the signal, and the duty cycle determines the ratio of the period during which the enable signal is high. Controlling the period and the duty cycle is essential as it introduces frequency components that render the voltage drop more significant due to the frequency-dependent PDN impedance [28].

The adversary must carefully control all the power wasters' variable parameters to ensure a substantial enough voltage drop to inject faults into the Trojan triggers path. The strength of the attack also affects how frequently the Trojan will be activated and, accordingly, how easy it will be to distinguish the leaked secret at the output, which is essential if the attacker cannot control the input. In addition, the adversary aims to limit the voltage drop to ensure the board does not reset.

Once the suitable parameters of the attack are identified, the victim's output can be collected to be analyzed. Prior to the attack, the adversary can collect a large set of outputs to obtain a baseline distribution of values at the output. Finally, the secret can be identified as (one of) the most repeated value(s) that appears only when the power wasters are active and never appears otherwise (i.e., a value that does not fit the baseline distribution). Optionally, if the victim exposes a public input interface to receive requests from other parties (e.g., the victim performs publicly available encryption services), the attacker can send a fixed set of predefined input values, observe the obtained outputs, and quickly distinguish the faulty output values and the leaked secret from the correct ones without having to examine the output distribution.

V. IMPLEMENTATION

We test X-Attack on an embedded FPGA platform and a commercial cloud FPGA instance. The embedded setup serves as a test for the attack feasibility and an initial exploration of its control parameters. Then, we port X-Attack on the cloud to understand how the differences between the two setups affect the attack parameters and to analyze in detail additional variables affecting the attack's success. In what follows, we describe the two experimental setups.

A. EMBEDDED SETUP

We use an Intel DE1-SoC development kit with Cyclone V SoC as the embedded test platform. Intel Quartus Standard Edition 19.1 is used for FPGA compilation [47]. Fig. 6 illustrates the system architecture. The FPGA provides a dual-core ARM Cortex-A9 as a hard processing system (HPS), which we leverage for control purposes. The design comprises the parts discussed in the following three sections: the attacker, the victim, and the data collection subsystem.

1) ATTACKER

The adversary employs power-wasting circuits to lower the chip's voltage, increase the path delays within the victim circuits, and inject faults. The goal with the embedded setup for X-Attack is to investigate the possibility of leveraging

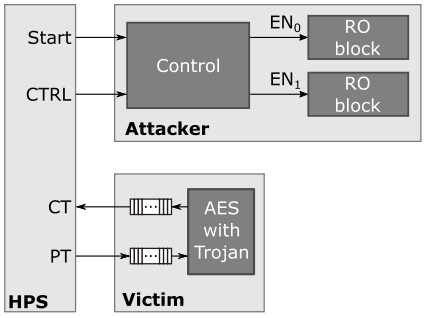


FIGURE 6. System design for the embedded setup.

the undervolting generated by the power wasters to activate the SDC Trojan. To that end, we use single-LUT ROs: they are effective and easy to control. Specifically, the attacker design consists of two large equally-sized blocks of ROs. The adversary can choose to continuously enable one block (resulting in only half of the ROs active), continuously enable both blocks (all of the ROs are active), or periodically enable and disable both blocks [41].

The software on the HPS controls the activation of the two RO blocks. The interface between the HPS and the FPGA allows the adversary to initiate the attack (`Start` signal in Fig. 6) and to choose how to activate the ROs (`CTRL` signal in Fig. 6). Based on `Start` and `CTRL`, the hardware generates the enable signals that control the ROs. Specifically, the `Start` initiates the attack. `CTRL` is a pair of two signals. If their values are `00` and `11`, then the power wasters are disabled and enabled, respectively. In our implementation, `01` defines the case when both blocks are toggling (i.e., periodically enabled and disabled), while `10` continuously activates only one block. The hardware terminates the attack once the preset duration has elapsed to ensure that FPGA-HPS communication delays do not cause the exploit to last longer than intended and risk resetting the FPGA. The clock frequency is 160 MHz, and the attack duration is set to 4,096 clock cycles (equivalent to 25.6 µs).

2) VICTIM

In a general context, an adversary would implant an SDC Trojan in an IP core with a secret to leak. Accordingly, and without loss of generality, our attack targets an AES cryptographic core. Symmetric encryption has two secrets: the plaintext and the encryption key. The adversary aims to obtain the key, which allows decrypting the ciphertexts.

For reproducibility of our results, we choose an open-source 128-bit pipelined AES implementation [48]. As the AES core is pipelined, targeting a specific round for a DFA exploit is challenging. X-Attack, on the other hand, does not attempt to gain information by introducing a fault into the algorithm. Instead, the exploit faults one or two specific paths (of the trigger signals) to leak the entire secret to the output of the infected IP, as illustrated in Fig. 5. The satisfiability analysis of Yosys [46] helps identify pairs of don't-care

signals within the Sbox, to use as Trojan triggers. We find that a suitable signal pair can be identified for any of the two-bit SDC conditions. However, we focus first on 11 (and, correspondingly, 00_{inv}), leaving a thorough analysis of other two-bit SDC conditions for the cloud FPGA setup.

Having identified a suitable signal pair with the SAT solver, we insert the Trojan shown in Fig. 5 (where dc_0 and dc_1 are the don't-care signal pair [24]) and verify that Intel Quartus does not optimize it away. Initial results show more frequent secret key leakage with 00_{inv} than with 11 SDC condition; hence, we opt for 00_{inv}. The delays of the corresponding don't-care signals as reported at the slow 100°C corner are 3.274 ns and 4.376 ns. From our experience, for the embedded setup, the slowest model is conservative enough to guarantee correct timing and aligns well with experimental observations: correct operation when no power wasters are active and faulty operation otherwise.

We supply the plaintexts to the AES in one of the following two ways: as the output of a 128-bit linear feedback shift register (LFSR) implemented in hardware [49] or a repeating sequence of two different plaintexts hardcoded in hardware. We shall refer to these options as *pseudorandom* and *fixed* plaintexts. The fixed plaintexts allow us to test the attack while knowing what each ciphertext should be. They also simulate cases where the AES encrypts plaintexts taking values within a limited range. The pseudorandom plaintexts correspond to the test case when the encryption requests arrive from various sources and take on values following a distribution unknown to the attacker. The key and the fixed plaintexts are provided in Appendix A.

3) DATA COLLECTION

To account for the clock frequency and throughput difference between the system's hardware and software counterparts, we use on-chip first-in, first-out (FIFO) buffers for data collection. While the exploit runs, the FIFOs store the ciphertexts and the corresponding trigger signal values. Although the adversary would generally have access only to the ciphertext, we store triggers to validate the results. The interface between the hardware and the software allows offloading of the collected values for processing by the HPS.

B. CLOUD SETUP

Both in deployed FPGA cloud platforms and in multitenant FPGA platforms proposed in the literature, the programmable fabric is divided into a static region (the shell) and a dynamic region for programming the user circuits [9], [50], [51]. The shell typically includes the memory and communication interfaces, and the FPGA tenants cannot modify it. Our implementation targets a commercial cloud FPGA instance providing access to Intel FPGA devices.

Intel cloud platforms use the Intel Open Programmable Acceleration Engine (OPAE) [52]. The OPAE is the software framework for managing and accessing FPGAs, providing users with a software development kit and the necessary Linux drivers. Within the Intel framework, the static region is called the FPGA Interface Manager (FIM), and it controls the communication between the FPGA and the host CPU through PCIe. The dynamic region, which includes supporting logic for interfacing with the host through the FIM, is called the Accelerator Functional Unit (AFU). The user needs to provide the register-transfer level (RTL) description of the function implemented by the AFU. The AFU is mapped to a partial reconfiguration region, where PLLs cannot be instantiated. Therefore, the user is constrained to selecting the desired clock frequency within the AFU specifications [53].

The AFU uses the Core Cache Interface (CCI-P) to communicate with the host. The CCI-P interface comprises three command/response channels: the AFU uses channels 0 and 1 for host memory read requests and responses and can also receive memory-mapped input/output (MMIO) read and write requests from the host on the response port of channel 0. In addition, channel 1 is used for issuing write fences and interrupts. Finally, the AFU uses channel 2 for sending MMIO read responses to the host [53]. The width of the data part of the interface is 512 bits [54].

We deploy the design for testing X-Attack on a rented commercially available cloud instance. Knowing that the CSP may attribute any available FPGA instance, we focus on one geographical region only where we counted seven different FPGA instances. We record the instance identification (ID) number in each experiment, to correctly aggregate the results corresponding to the same FPGA instance. For comparison purposes, we repeat some experiments on more than one FPGA instance. The results we will present in Section VI will target the same instance unless noted otherwise; yet, we will show and discuss if the observed effects apply to other tested FPGAs. The FPGA instances are Intel Arria 10 GX programmable accelerator cards (PACs) [29]. Our design follows the platform structure of the Intel OPAE framework. The system design for the cloud setup is similar to the embedded setup, with changes only to adapt to different interfaces and ensure the AFU requirements are met. It comprises the three main parts shown in Fig. 7: the attacker, the victim, and the communication and control interfaces. The CSP provides the shell design and functionality, which we use as is.

1) ATTACKER

As the targeted cloud instance does not enforce any constraints preventing potentially malicious constructs (e.g., combinational loops), we opt for power wasters that are effective and easy to control: ROs, grouped in blocks of $N_W = 4{,}096$ instances each. We found experimentally that this number is a good compromise between having an overly complex control network (i.e., too many enable signals) and overly sized RO blocks (and, consequently, large voltage steps). In total, $N_B = 20$ blocks are instantiated, which use about 10% of the FPGA resources. More than 20 active RO blocks are likely to cause FPGA reset. The RO blocks are

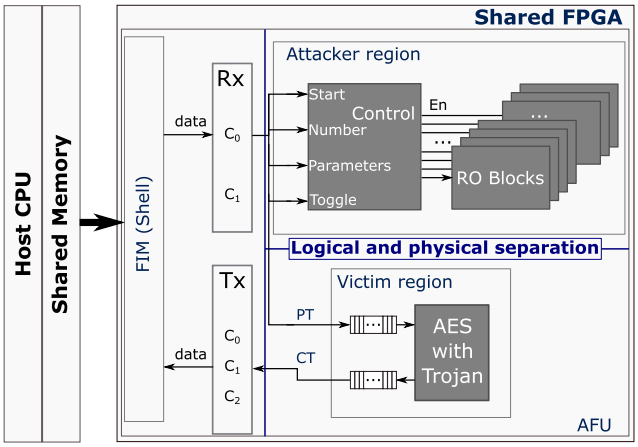


FIGURE 7. System design for the cloud setup.

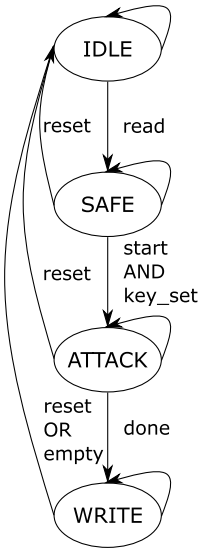


FIGURE 8. Finite state machine used for the interface between the host CPU and the user region in the FPGA.

constrained to a LogicLock region in one half of the FPGA (Fig. 13). The control interface is configured to allow the user to choose the number of RO blocks to activate in each experimental run. The clock frequency is 320 MHz. The attack typically lasts 4,096 clock cycles, equivalent to 12.8 µs. The user can set the frequency and the duty cycle of the RO enable signal.

2) VICTIM

The same victim is used as in the embedded setup: the open-source pipelined AES core [48]. We constrain the placement of the victim to a LogicLock region in the remaining half of the FPGA (Fig. 13); the adversary and the victim are logically and physically separated. The AES operates at 320 MHz. We validate that all obtained ciphertexts are correct and that no key leakage occurs in the absence of RO activity.

Two sets of experiments are performed. The first is a detailed analysis of factors impacting the attack's success. To collect a statistically relevant number of successful attack attempts for such an analysis, we increase the delay of the original don't-care signal pair (dc_0 and dc_1). Specifically, we add buffers to the signal paths and apply the transformations detailed in Section IV. As a result, the trigger signals delays for the SDC condition `11` as reported at the fast 100°C corner are 2.784 ns and 2.699 ns. The second set of experiments validates the attack feasibility and measures the success rate when the trigger signal delays are unmodified (see Section VI-D).

Similarly to the embedded setup, we supply the plaintexts to the AES either as the output of a 128-bit LFSR implemented in hardware [49] or a sequence of 16 hardcoded predefined plaintexts. The secret key and the sequence of fixed plaintexts are listed in Appendix A.

3) COMMUNICATION AND CONTROL

The host CPU provided with the cloud instance controls the functionality of the FPGA by supplying all necessary parameters. We follow the sample design provided by Intel and modify it to integrate the desired functionality of our setup [55]. The implemented interface is shared between the victim and the attacker for experimental purposes. The sharing makes synchronization easier but does not affect the exploit's success, as synchronization can alternatively be achieved through side channels [27], [56]. The main requirement is for the power wasters to be active while the victim runs. Similarly to the embedded design, we collect all data during an experimental run in on-chip FIFOs. After the run terminates, the hardware offloads the data to the host CPU of the cloud instance. The FIFOs operate at a fixed frequency of 160 MHz, half of the frequency of the AES. We collect the data in two FIFOs to avoid timing violations when writing to the buffers.

Following the example design from Intel, the finite state machine (FSM), shown in Fig. 8, controls the user region. The FSM always starts in the `IDLE` state, where all parameters are reset to their default value. After the host de-asserts the reset signal, the interface waits for a valid signal on the receiver of channel 0 to read the parameters specified by the host CPU. These parameters include the enable signal frequency and duty cycle. Once the interface has read these parameters, it asserts the `read` signal. Once `read` is asserted, the FSM transitions into the `SAFE` state. Within the `SAFE` state, if the victim was reset, it can operate freely without an active attack. This window of safe operation allows the key of the AES to also be set (asserting the `key_set` signal). During the `SAFE` state, the interface waits for the control signal, which determines the number of power wasters to activate and their activation pattern, and for the `start` signal. Once `start` is asserted by the host CPU, and the victim is operating with a valid key, the attack commences, and the FSM reaches the `ATTACK` state. The FSM then awaits the assertion of the

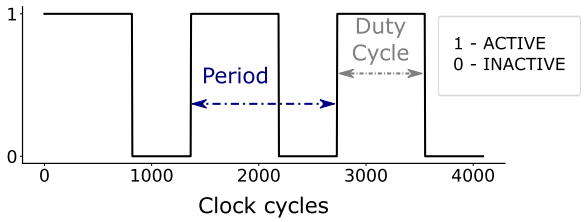


FIGURE 9. Enable signal and the corresponding set of parameters.

done signal to indicate that the attack has finished and to transition into the WRITE state. Finally, the interface controls the transfer of the data collected during the exploit to the host CPU in the WRITE state and stops once the FIFOs that collected the data are empty. If the reset signal is asserted during any state, the FSM returns to the IDLE state. If the transition condition is not asserted, the FSM remains in its current state.

VI. EXPERIMENTAL EVALUATION

This section presents the experiments we carried out on embedded and cloud FPGAs to evaluate the success of X-Attack and the factors affecting it. The first set of experiments focuses on the adversary, with the goal of identifying the optimal parameters of the power-wasting circuits. Then, we turn to the victim to examine how the different SDC conditions for the hardware Trojan affect the exploit's success. We then investigate the effects of changing the distance between the attacker and the victim. Finally, we examine the success of the exploit when the trigger signal delays are decreased to avoid the critical paths of the victim IP.

A. FINDING OPTIMAL ATTACKER PARAMETERS

The adversary employs the power-wasting circuits with the goal of consuming enough power to increase the signal delays to the point of activating the SDC Trojan but, ideally, not causing faults in the communication interfaces or the reset of the board. To achieve that, we give the attacker control over the attack duration and the following set of parameters for tuning the voltage drop (see Fig. 9):

- Attacker size: the number of ROs used in the attack, or the number of blocks receiving the enable signal.
- Attacker mode: whether the enable signal of the power wasters is constantly active or toggling for the entire attack duration (the latter corresponds to the scenario shown in Fig. 9).
- Enable signal period: the number of clock cycles to complete the pattern of activating the power wasters before repeating it again.
- Enable signal duty cycle: the ratio of the number of clock cycles for which the enable signal is active over the enable signal period.

1) ENABLE SIGNAL PARAMETERS

Embedded Setup: Finding the RO enable signal parameters for the embedded design did not require sweeping a large set of values. The parameters were easy to find thanks to the similarity between ours and the setups in previous works on remote fault injection [18], [22], [41]. First, both blocks of ROs were activated with a toggling enable signal; the enable signal's period and duty cycle were set to 400 ns and 75%, respectively. Then, one block was disabled, and one was continuously enabled. Finally, both blocks were continuously enabled. The software controls the sequence of activation patterns, targeting a toggling enable signal for the first 48% of the attack duration, followed by a short activation of one block, and then the remaining half of the attack duration is spent with both blocks continuously active.

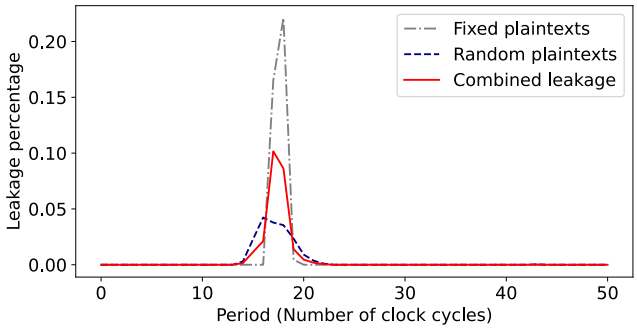


FIGURE 10. The average occurrence of leakage (i.e., successful Trojan triggering over the total number of attempts) in percent, as a function of the period of the RO enable signal. A relatively small attacker containing seven RO blocks (28,672 ROs) was instantiated.

Cloud Setup: In the cloud instance, the PDN differs, and, in the literature, there are no results for fault injection on the same platform on which to base the choice of enable signal parameters. Therefore, we swept the period and duty cycle of the enable signal. Additionally, we gradually increased the power-wasting effects to avoid resetting the remote FPGA. Once we observed the reset of the data collection FIFOs, while the board *remained responsive*, we considered the attack to be too strong, and stopped increasing the number of power wasters. Regarding the RO enable signal pattern, we found that continuously activating the ROs is ineffective for fault injection. Toggling the activation of the power wasters is necessary to successfully inject faults, which aligns with observations in previous work [18], [22], [28]. Comparing continuous toggling, on one side, with toggling for the first half of the exploit followed by continuous activation in the second half, on the other, we observed that the latter configuration increased the likelihood of the FIFOs resetting. Consequently, we opted for toggling the enable signal for the entire duration of the exploit.

Next, we focused on the duty cycle sweep. The initial experiments revealed that a duty cycle above 50% is more likely to affect the FIFOs, increasing the risk of a DoS attack. Hence, we keep the duty cycle around 30%. To minimize the risk of a DoS, a relatively small attacker containing seven RO blocks (28,672 ROs) was instantiated.

Fig. 10 presents the results of the sweep of the enable signal period. For each configuration, the ratio (in percent) of AES encryptions where the Trojan was successfully triggered

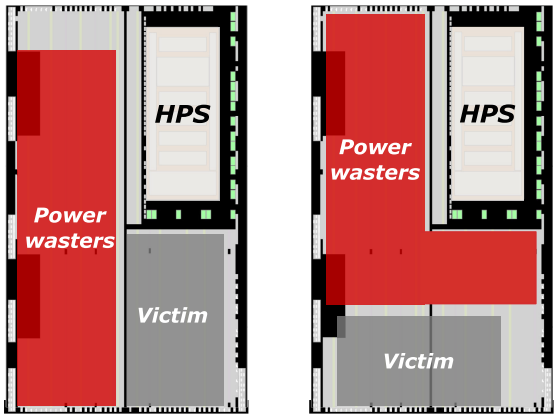
over the total number of encryptions is computed. The three lines plotted correspond to fixed plaintexts, pseudorandom plaintexts, and the sum (i.e., the total or combined leakage). We report the average over five experiments, each containing 15 runs of 4,096 encryptions of fixed plaintexts and 15 runs of pseudorandom plaintexts. The results indicate that the leakage occurs when the enable signal period is between 14 (∼23 MHz) and 22 clock cycles (∼14 MHz). For a larger attacker (typically 12 blocks or more), the faulting range is wider, reaching 45 MHz. The obtained results are consistent with previous work, which has shown that frequencies close to the PDN resonance frequencies lead to stronger power-wasting effects [28]. Another interesting observation from Fig. 10 is that the probability of triggering the Trojan changes with the choice of input plaintexts; we will return to these effects in greater detail in Section VI-A2.

2) ATTACKER SIZE

The power wasters' activity effects depend on the enable signal parameters and the resources used. We expect the power consumption to increase with the number of power wasters. Yet, the optimal attacker size (i.e., the size resulting in frequent leakage) is likely not the maximum possible because too large an attacker can cause faults other than those required to activate the Trojan (e.g., by inducing more bit flips than needed or by faulting the communication interfaces, FIFOs, etc.). To find the suitable attacker size, experiments are carried out as follows.

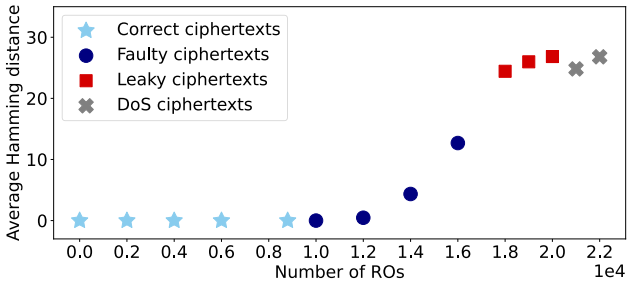
Embedded Setup: We vary the total number of ROs for two design floorplans shown in Fig. 11. In the first, shown in Fig. 11a, the attacker is in a LogicLock region on the left side of the FPGA, and the victim is constrained to the right, while in the second, shown in Fig. 11b, the attacker is in the space above the victim. The Trojan is embedded into the AES circuit with SDC condition 00_{inv} without any additional delay (as explained in Section V-A2). The baseline is the case where no ROs are active. The attacker's region is then gradually filled with ROs; the maximum number of instantiated ROs was 22 k (34.3% of the total number of adaptive logic modules (ALMs) available). The trigger signal delays were 3.274 ns and 4.376 ns for the floorplan in Fig. 11a, and 2.639 ns and 4.17 ns otherwise.

In the initial experiments (i.e., with small attacker sizes), we send fixed plaintexts to the AES. Knowing the expected ciphertexts permits detecting the first occurrence of faulty outputs, and documenting the Hamming distance (HD) between the observed ciphertexts and their expected values. For the attacker sizes for which we observe faults, we supply the pseudorandom plaintexts, to document how often the secret key leaks to the output (i.e., the attack is successful) and whether the FPGA resets. Fig. 12 summarizes the results for the two floorplans and 100 repetitions of the experiments. Four different symbols highlight four attack outcomes: the AES operation was correct, faults other than the desired occurred, the secret leaked (i.e., the Trojan was triggered), and the board reset. The y-axis is the HD, i.e., the number

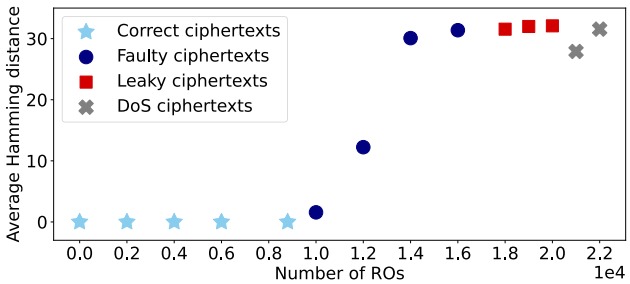


(a) Attacker on the left (b) Attacker on the top

FIGURE 11. Floorplans in the embedded setup experiments.



(a) Attacker on the left



(b) Attacker on the top

FIGURE 12. Hamming distance between the obtained and the expected ciphertexts, averaged over 100 runs.

of faulty bits in the ciphertext. The attacker size for which the Hamming distance becomes higher than zero marks the beginning of the appearance of faults. There is clearly a range of attacker sizes suitable for fault injection.

Given that the increased power consumption leads to a lower voltage and increased on-chip delays, we expect to see effects similar to increasing the clock frequency and inducing timing faults. Consistently with the expected behavior, Fig. 12 shows faults occur before the key leakage. As the Trojan triggers are not critical paths of the AES design, the high-criticality paths fail first. As the number of ROs increases, more faults occur, as the growing HD indicates. Again, this result matches the effect that we expect an even higher clock frequency to have. Then, at 18k ROs, with

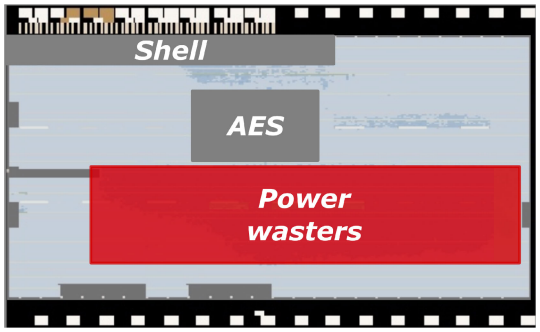


FIGURE 13. Floorplan in the cloud setup. The view is rotated by 90° for space reasons.

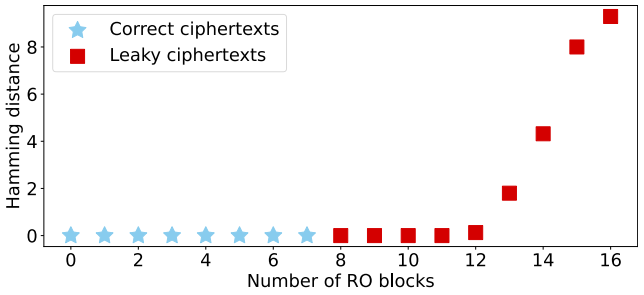


FIGURE 14. HD between the obtained and the expected AES ciphertexts for the fixed plaintexts case, averaged over 10 experiments, each containing 15 runs with fixed plaintexts and 15 runs with pseudorandom plaintexts; a run consists of 4,096 encryptions.

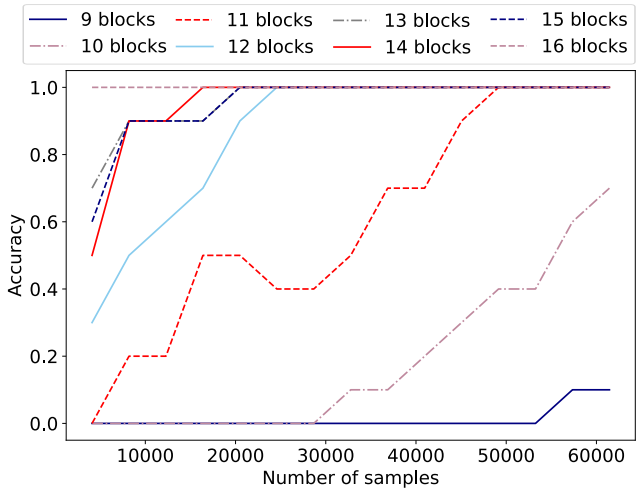


FIGURE 15. The probability that the key is the most occurring output value, computed as a function of the number of obtained ciphertexts (i.e., output samples) and attacker size. We refer to this probability as the accuracy of the prediction that the key is the most occurring output value.

faults still occurring, the attack triggers the Trojan. Finally, with more than 20k ROs, the board resets before the end of the 100 runs of the attack. From the results, it is clear that the attacker size is a factor the adversary needs to carefully control to ensure being within the range that causes leakage but not reset. It is worth noting that the two different RO placements, while affecting the faults observed (as seen by the different HD), do not significantly affect the adversary's ability to distinguish the key as the most repeating value at the output [22].

Cloud Setup: Turning to the cloud setup and a larger FPGA, we increase the attacker size to 20 blocks, each with 4,096 ROs. Activating an attacker size larger than 16 blocks results in the FIFO resetting (the exact number of ROs that trigger the FIFO reset varies slightly from one FPGA instance to another), so we limit our analysis to 16 blocks and leave the last four implemented blocks unused. The period of the enable signal and the duty cycle are set to eight clock cycles and 62.5%, respectively. We set the period to a conservative value (below the range that caused leakage in Fig. 10) to avoid resetting the FIFOs or the FPGA. The somewhat higher duty cycle is chosen to compensate for the lower period. Fig. 13 illustrates the floorplan, where the RO blocks are constrained to a LogicLock region in one half of the FPGA.

First, we examine if the attack causes faulty AES encryptions or the leakage of the key to the output. We find the HD between the obtained and expected ciphertexts in the function of the number of active attacker blocks for the fixed plaintexts case. To collect data, we run ten experiments, each containing 15 runs with fixed and 15 runs with pseudorandom plaintexts; a run consists of 4,096 encryptions. Fig. 14 shows how the HD between the obtained and expected ciphertexts, averaged across the ten experiments, changes with the number of active attacker blocks. As expected, the HD increases with the attacker size because the number of faults is likely to increase when a more aggressive attacker is active. Leakage starts occurring when eight RO blocks (32,768 ROs) are active. These trends are similar to the embedded setup (Fig. 12). The main difference is the minimum attacker size required for fault injection: it is larger for the cloud setup, which is expected because cloud FPGA instances have a higher-quality PDN. Finally, we observe no DoS, which is exactly what an adversary would want to achieve.

The next important insight is the relationship between the attacker size and the number of attack runs (determining the necessary total duration of the attack), on one side, and the probability that the secret key is the most frequently appearing value at the victim output, on the other. For a predefined attacker size, we run ten experiments, each with 15 attack runs of 4,096 encryptions of pseudorandom plaintexts (61,440 encryptions per experiment). In every experiment, we take N_{CT} ciphertexts ($0 \leq N_{CT} \leq 61,440$) and count the number of occurrences of the secret key. If the key was the most frequently encountered value among the analyzed ciphertexts, we consider that attack duration sufficient for a successful attack. We count in how many (out of 10) attacks the duration was sufficient to compute the final success rate. We see the obtained success rate as an estimate of the *accuracy* of the prediction that, for a certain number of encryptions, the most repeating value at the output will be the leaked secret. Fig. 15 summarizes the results for attacker sizes ranging from nine to 16 blocks. For

example, looking at the plot corresponding to the attacker size of nine blocks, we see that at least 58k ciphertexts (i.e., 14 attack runs) were required for the key to become the most frequently appearing value among the faulty ciphertexts. Furthermore, it was only in one of the ten experiments that the above was observed (hence the likelihood/accuracy of 0.1, i.e., 10%). From Fig. 15, we conclude that the key becomes the most occurring value (with 100% probability) when the adversary employs 11 or more RO blocks. The attack is more successful earlier for larger attacker sizes; hence, if the duration of the attack is of concern, an adversary should deploy more ROs. Interestingly, the correlation between the attacker size and the accuracy is not always consistent, e.g., an attack with 14 blocks attains 100% accuracy earlier than with 15 blocks. We will explain shortly why a stronger attack can result in undesired faults, effectively reducing the likelihood of the leakage. In conclusion, with enough ciphertexts (output samples) collected and enough power wasters, the key eventually becomes the most repeating output value. It is worth noting that the key search space is significantly reduced if the adversary considers the top few (e.g., top 10) most occurring values at the output.

Finally, it is necessary to investigate the factors that impact the observed leakage. Fig. 16 shows the average absolute number of recorded occurrences of leakage in the function of the number of RO blocks, for the fixed and the pseudorandom plaintexts in isolation, as well as the corresponding sum. As observed earlier, the leakage increases with the attacker size. However, in the fixed plaintext case, the leakage appears to decrease when the attacker size grows beyond 15 blocks. Given that for that number of ROs, the attack is not yet strong enough to cause a reset, the results seem counterintuitive. To understand why, we analyze the Trojan triggers (i.e., the don't-care signal pair after the transformation in Fig. 5) before and during the attack more closely in Table 1. The rows are the values of the trigger signals in the absence of the attack—the SDC condition `11` is omitted as it normally never occurs in the absence of the attack; the columns show the observed faulty values. For each trigger signal pair, the average count of occurrences of the faulty values is given. As no change in the trigger equals no fault, the corresponding cells are left empty (symbol -). The first column of Table 1 specifies the number of active RO blocks.

The results in Table 1 suggest that the trigger signal combinations (which depend on the workload) and the observed faults determine how often the key will leak (i.e., how often the desired SDC condition is reached). The first notable behavior is that, regardless of the attacker size, only the value `10` succeeded in faulting to `11` (as highlighted in bold in Table 1). Examining the average occurrence rate of the faults from `10` to `11`, we see that increasing the attacker size, in general, results in more leakage. An exception is the transition from 15 to 16 blocks, where the leakage decreases. The explanation is quite simple, in fact. Looking closer at the total number of faults occurring for the trigger `10` (i.e., the *sum* of all the values in the corresponding rows), we find

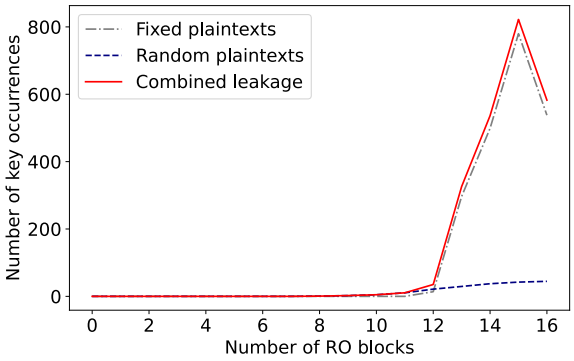


FIGURE 16. Number of occurrences of the key at the output, averaged over 10 experiments, each containing 15 runs with fixed plaintexts and 15 runs with pseudorandom plaintexts; a run consists of 4,096 encryptions.

TABLE 1. Summary of the average number of faults observed for the 11 SDC condition with various attacker sizes for the fixed plaintexts case. The numbers are averaged over ten experiments each containing 15 runs. One run consists of 4,096 encryptions. In bold, the faults leading to leakage. In italic, the values corresponding to double bit flips.

Blocks		`00`	`01`	`10`	`11`
12	`00`	-	0.06	0	**0**
	`01`	1.07	-	*0*	**0**
	`10`	0	*0*	-	**0.92**
13	`00`	-	14.63	0	**0**
	`01`	30.22	-	*0*	**0**
	`10`	0.01	*0*	-	**19.78**
14	`00`	-	47.79	0	**0**
	`01`	63.53	-	*0*	**0**
	`10`	0.97	*0.01*	-	**33.26**
15	`00`	-	165.51	0.01	**0**
	`01`	88.43	-	*0*	**0**
	`10`	14.41	*12.57*	-	**51.99**
16	`00`	-	177.25	2.29	**0**
	`01`	122.95	-	*0*	**0**
	`10`	2.84	*47.46*	-	**35.86**

that with the attacker size growing, more faults do occur. However, as the attack grows stronger, the probability of other faulty values increases: for 12 blocks, only one type of fault is observed: `10` faults to `11` (one bit flip, from `0` to `1`). When the attack becomes stronger, `10` starts faulting to `00` and `01` (two bit flips); these faults are not useful to the attacker, as only reaching the desired SDC condition matters for the Trojan to leak the secret. The appearance of two bit-flips indicates that the attack is stronger than needed, reducing instead of improving the overall attack success.

The above-discussed effects likely differ when pseudorandom (or, simply, unknown) plaintexts are used because of the different probability distribution of the trigger signal pair values. Indeed, Fig. 16 shows a different leakage trend when testing with pseudorandom plaintexts: the number of leaky samples increases at a slower pace. For a pseudorandom (or, in a general case, unknown) sequence of plaintexts, the value `10` may occur infrequently, reducing the likelihood of

leakage and of double bit-flips for the given attacker size. To overcome this challenge, the attacker has at least two options. First, if some knowledge of the workload is available (e.g., their probability distribution), the adversary can use it to calibrate the attack strength so as to improve the likelihood of the successful exploit. Or, if the attacker has control over the victim inputs, they can repeat the attack with various input sequences to find one that increases the likelihood of leakage for the given number of RO blocks.

B. EFFECTS OF SDC ON ATTACK SUCCESS

Previous experiments on the cloud FPGA instances considered SDC condition `11`. In a general case, when trying to insert an SDC hardware Trojan, one may find a variety of candidate don't-care signal pairs with different SDC conditions. Furthermore, choosing one condition over the others may matter. In this section, we investigate in what way the choice of the trigger signals and their SDC condition affects the exploit's success. We hypothesize that the associated signal delays and the likelihood of the normally occurring values faulting precisely to the targeted SDC condition (governed by both the choice of the trigger signals and the input plaintexts) are the main factors impacting the attack's success.

1) TRIGGER SIGNALS

To answer the above hypothesis, repeating the experiments with a range of valid SDC conditions is required. With Yosys [46], we find signal pairs for SDC conditions `00`, `01` and `11`, while ensuring that every pair can reach all combinations of two-bit values except the SDC condition itself. To the above SDC conditions we further add $\texttt{11}_{inv}$, $\texttt{10}_{inv}$, and $\texttt{00}_{inv}$, by inverting the original three. As a result, we obtain six slightly different variants of the AES with the SDC Trojan (the difference being the choice and routing of the trigger signal pair).

2) DISTRIBUTION OF TRIGGER VALUES

As described in Section V-B2, depending on the experiment, we supply the plaintexts to the AES in one or both of the following two ways: a fixed sequence of 16 plaintexts (listed in Appendix A) or a sequence of pseudorandom numbers generated with a 128-bit LFSR implemented in hardware. When using fixed plaintexts, we know the expected ciphertexts. Hence, after the attack, we are able to tell exactly which ones are faulty and which bits are faulty. On the contrary, with pseudorandom plaintexts, we cannot tell if a ciphertext is faulty; we can only measure the distribution of the values at the output and look for outliers. This scenario corresponds to a case when adversary has no control over the victim inputs.

We know that the values the trigger signal pairs take during encryption depend on the plaintexts. Therefore, we hypothesize that the likelihood and nature of faults (i.e., which bit flips occur and with what probability) also depend on the plaintexts. To evaluate this hypothesis, we start by running encryptions and recording the trigger signals.

TABLE 2. Distribution of the Trojan trigger values (in %) for all tested SDC conditions in the absence of the attack. AES encrypts *pseudorandomly* chosen plaintexts. Each row corresponds to a different SDC condition, i.e., a different trigger signal pair. The experimental runs do not always use the same sequence of pseudorandom plaintexts, due to delays in CPU-to-FPGA communications. Accordingly, the distributions are not exactly the same for an SDC condition and its inverse.

SDC	00	01	10	11
00	0	47.52	2.33	50.15
$\texttt{00}_{inv}$	0	0.39	49.64	49.98
01	50.03	0	46.09	3.88
$\texttt{10}_{inv}$	3.88	46.08	0	50.03
11	49.94	49.67	0.39	0
$\texttt{11}_{inv}$	49.95	2.34	47.70	0

TABLE 3. Distribution of the Trojan trigger values (in %) for all tested SDC conditions in the absence of the attack. AES encrypts a *fixed set* of 16 plaintexts (Appendix A). Each row corresponds to a different SDC condition, i.e., a different trigger signal pair.

SDC	00	01	10	11
00	0	62.5	0	37.5
$\texttt{00}_{inv}$	0	6.25	31.25	62.5
01	50	0	50	0
$\texttt{10}_{inv}$	0	50	0	50
11	62.5	31.25	6.25	0
$\texttt{11}_{inv}$	37.5	0	62.5	0

Table 2 shows the distribution of the values of the trigger signal pair, for all tested SDC conditions and for pseudorandom plaintexts. The first column lists the SDC conditions, each corresponding to a specific trigger signal pair. The next four columns show how often (in %), in the absence of the attack, the trigger signal pair takes the values `00`, `01`, `10`, and `11`. Table 3 shows how the distributions change when pseudorandom plaintexts are replaced with a fixed set of 16 plaintexts (listed in Appendix A).

Tables 2 and 3 confirm that the distributions differ, as expected. Moreover, some trigger combinations are significantly more frequent than others, which we expect will significantly impact the likelihood and the nature of faults.

3) ANALYSIS OF THE FAULTS

To each SDC pair, we add delay elements (buffers alone or buffers in combination with transformations in equations 1, 2, and 4) to, first, equalize the SDC signal delays (preventing delay imbalance from impacting the conclusions) and, second, make them the victim's critical path. The consequence of the latter is that a statistically significant number of trigger events should occur, allowing us to perform insightful statistical analysis. The baseline experiments run without ROs active and with fixed and pseudorandom plaintexts. Next, we activate the ROs with the same enable signal

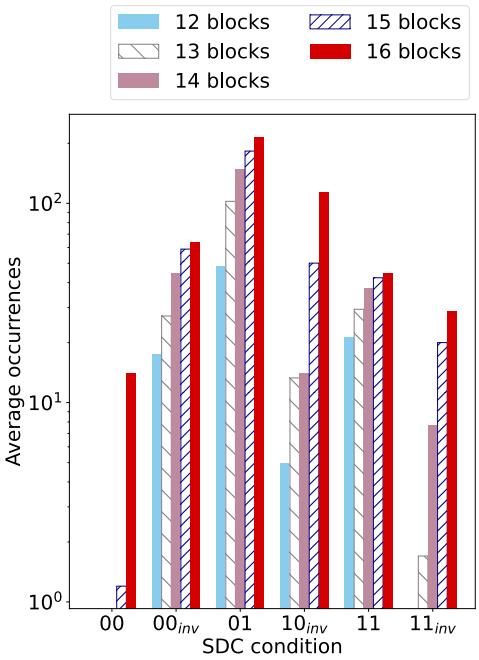


FIGURE 17. Average leakage occurrences (log-scale) for each tested SDC condition, multiple attacker sizes, and pseudorandom plaintexts.

TABLE 4. Delays (in nanoseconds) of the trigger signals for each tested SDC condition.

SDC	t_1	t_0
00	2.758	2.637
00_{inv}	2.819	2.834
01	2.566	2.721
10_{inv}	2.542	2.755
11	2.699	2.784
11_{inv}	2.744	2.709

parameters as before (period of eight clock cycles and a duty cycle of 62.5%), which should guarantee a successful attack without causing a DoS.

Fig. 17 reports the average number of successful attacks (i.e., a desired fault occurring) as a function of the chosen SDC condition, for multiple attacker sizes and pseudorandom plaintexts, for one FPGA instance. We include the results for another instance in Appendix B. Not unexpectedly, leakage occurs for every tested SDC condition. Most interestingly, the highest number of leakage events corresponds to the trigger condition 01. In comparison, the SDC conditions 00 and its inverse 11_{inv} are triggered least often. The similar likelihood of faults for 00 and 11_{inv} is due to our inverting 00 to obtain 11_{inv}, which resulted in keeping the distribution of trigger signal values the same for both SDC conditions.

To understand the reasons for the results in Fig. 17, we first extract the delays of the don't-care signals from Intel Quartus. In Table 4, t_1 and t_0 are the most- and least-significant bit of

TABLE 5. Summary of the analysis of the likely faults and the faults resulting in leakage for the tested SDC conditions.

SDC	Slow	Most occurring	Likely faults
00	t_1	11, 01	11 → 01; 01 → 11
00_{inv}	t_0	11, 10	11 → 10; 10 → 11
01	t_0	00, 10	00 → 01; 10 → 11
10_{inv}	t_0	11, 01	11 → 10; 01 → 00
11	t_0	00, 01	00 → 01; 01 → 00
11_{inv}	t_1	10, 00	10 → 00; 00 → 01

the trigger signal pair, respectively. We observe that the delays of the trigger signals do not necessarily correlate with the key leakage. The SDC condition 00_{inv} has the highest delays of the six conditions tested, as seen in Table 4. However, the highest leakage occurs for the 01 case, according to Fig. 17. The SDC condition 00_{inv}, with the pseudorandom plaintexts, has a distribution of the trigger signal values where 01 appears relatively infrequently (0.39% according to Table 2). With the least significant bit being slower (third row in Table 4) and, hence, more vulnerable to fault injection, 01 is the value that would most likely fault to the SDC condition. Given that 01 occurs infrequently compared to the other values, the likelihood of a fault leading to the SDC condition 00_{inv} is significantly reduced. We carry out a similar analysis for the other SDC conditions and summarize it in Table 5. The situation is different for the SDC condition 01, where the value 00 occurs frequently when using the pseudorandom plaintexts, according to Table 2. As t_0 is slower than t_1 (Slow column in Table 5), it is more susceptible to fault (fourth row in Table 4) and, hence, 00 faulting to 01 is a likely event, as highlighted in the fourth row in Table 5. The trigger signal faults *likely to occur* result from a fault affecting the *slower* of the two signals, in the *most frequently* occurring pairs of values. Accordingly, considering the example of SDC condition 01, the faults most likely to occur are 00 (the most occurring column in Table 5) faulting to 01 (the likely faults column in Table 5) and, similarly, 10 faulting to 11. One can reason similarly about the other SDC conditions. We will shortly ascertain that the distribution of the input plaintexts affects the resulting faults and leakage, when we examine the case with the fixed plaintexts.

To check whether the predicted faults correlate with the experiments, we use fixed plaintexts (because the correct values of the trigger signals are known). Table 6 shows the number of observed faults for each trigger signal combination. The list of triggers and faulty values are given in the second column and the first row, respectively. Some trigger signals in our implementation never occur in the fixed plaintext case; those are marked in italic. Some values not occurring means that all the faults that can be obtained by flipping one of the trigger signals are equally unlikely. Given the differences between the fixed and pseudorandom input cases, the results in Table 6 do not necessarily match the observations in Fig. 17. However, they give us an indication

TABLE 6. Average number of occurrences of the faults in the trigger signals for all tested SDC conditions over ten experiments each containing 15 runs with fixed plaintexts. The attack uses 16 blocks of ROs. The values in bold denote the number of occurrences of the bit flips necessary for leakage. The values in italic indicate that the original value does not occur, and so all transitions in that row never happen.

SDC		00	01	10	11
00	01	**0**	-	0	22.4
	10	**0**	*0*	-	*0*
	11	**0**	19.4	0	-
00_{inv}	01	**0**	-	28.45	60.37
	10	**0**	0.01	-	22.47
	11	**0**	0	76	-
01	00	-	**0**	0	0
	10	0	**0**	0	0
	11	*0*	**0**	*0*	-
10_{inv}	*00*	-	*0*	**0**	*0*
	01	22.9	-	**0**	259.12
	11	11.32	237.48	**0**	-
11	00	-	177.25	2.29	**0**
	01	122.95	-	0	**0**
	10	2.84	47.46	-	**35.86**
11_{inv}	00	-	0	6.85	**0**
	01	*0*	-	*0*	**0**
	10	2.21	0	-	**0**

and allow us to analyze the effect of the SDC condition and the distribution of the trigger signal values on the exploit's success. Moreover, the differences across various SDC conditions present useful insights into the fault-injection exploit.

Examining Tables 5 and 6 together, we see that the predicted faults based on the slowest trigger signal and the most occurring trigger signal values match with the results in Table 6. For example, for SDC condition 00, Table 5 predicts most likely faults to be 01 and 11. This behavior is confirmed in Table 6, where the three rows corresponding to the SDC condition 00 show only 19 occurrences of 01 and 22 occurrences of 11. Somewhat different, yet explainable, effects are seen with 01 and 10_{inv}. For these two cases, the trigger signal with the largest delay does not toggle, due to one of the possible trigger signals values not occurring with the fixed plaintexts (e.g., 00 for SDC condition 10_{inv}, marked by the italic in Table 6). As a result, the one signal which does not toggle (in these two cases being the *slow* trigger signal) faults with a very low probability, and hence, faults that are observed are different than what would occur if the *slow* trigger signal had faulted. In fact, the issue is not the attack strength as the attack can induce faults in the AES IP (we do observe faulty ciphertexts that are not the key for the 01 case). At the same time, the values of the trigger signals specifically are not faulty, supporting the hypothesis that whether or not a signal switches affects the likelihood of the attack faulting that signal.

Finally, we notice the occurrence of two-bit faults for the cases 10_{inv}, 11 and 00_{inv}, as shown in Table 6. While Fig. 17, which corresponds to the pseudorandom inputs case, does not show any decrease in the leakage, the two-bit faults observed here, for the fixed plaintext case, could indicate that the attack is stronger than necessary. This observation is consistent with the results in Fig. 16, which showed a decrease in the leakage for the fixed plaintexts for the SDC condition 11.

In conclusion, the distribution of plaintexts, the trigger signal values, the possibility of the trigger signals switching (affected by their distribution), and the delays of the trigger signals all affect the leakage rate and the success of the exploit. Therefore, while the adversary inserting the Trojan can choose the trigger signals and their SDC condition, the success of X-Attack will also depend on the input distribution, which may or may not be controlled by the adversary, and how the inputs interact with the key to produce the internal signals that are used as trigger signals (potentially changing the ability of the trigger signals to switch and accordingly the likelihood of the undervolting affecting them).

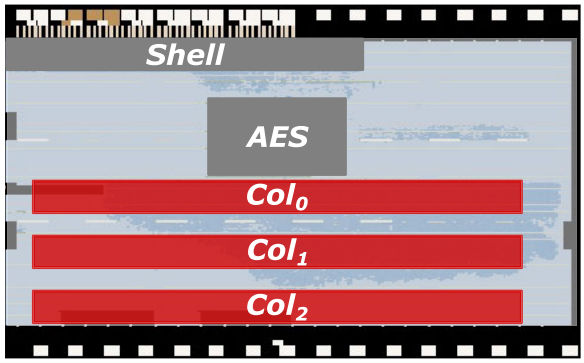


FIGURE 18. Floorplan for testing the effect of the distance on the success of X-Attack in the cloud. For space reasons, the view is rotated by 90°.

C. EFFECT OF DISTANCE ON ATTACK SUCCESS

With the optimal attack parameters chosen for a specific Trojan-infected victim, the adversary then tries to ensure that the victim is colocated on the same FPGA. While the attacker can try to identify cotenants and ensure cotenancy with the victim, only the CSP can control the distance between the tenants. The power-wasting effect propagates throughout the chip and can even affect components not within the programmable logic [16]. However, the attack has the strongest effect on the logic closest to the power wasters. As a result, the distance between the power wasters and the Trojan can require the adversary to change the attack parameters.

We test the effect of the distance on the success of X-Attack by placing the attacker as shown in Fig. 18. Each column consists of seven blocks (for a total of 21 blocks), where a block has 4,096 ROs. We can activate each column separately, in pairs, or all together. Since a column has only seven RO blocks, activating one at a time results in no leakage. However, activating them all results in a voltage drop that is too strong. The effect of the distance is apparent when activating Col_0 and Col_1 together compared to activating Col_1 and Col_2. For 150 runs with fixed plaintexts and

150 with pseudorandom plaintexts, the average number of leakage occurrences is two for Col_0 and Col_1, while it is 0.6 for Col_1 and Col_2. Faults occur more often, with approximately 16 on average for the closer placement and 15 for the farther. The distance appears to affect the leakage more than the occurrence of faults, likely because leakage requires faulting only one or two specific paths (of the trigger signals). In comparison, faulting higher criticality paths is more likely, leading to faults other than the secret key leakage. We observe the same effects when repeating the experiment on a different FPGA instance. In conclusion, the larger the separation between the attacker and the victim, the more challenging it is for the exploit to succeed.

D. RUNNING X-ATTACK

The experiments in the previous sections studied the correlation between attacker parameters, Trojan parameters, and deployment parameters, on one side, and the key leakage, on the other side. As mentioned earlier, to enable the collection of statistically significant number of faults, the trigger signals delays were increased through the transformations in Section IV-A, to the point of becoming the critical path of the victim design. However, such implementation may render the Trojan easier to detect (e.g., if one decides to inspect the signals on the critical path). Our final experiment aims to assess the attack's success in a different, stealthier setup: on the cloud, with trigger signals not on the critical path.

To that end, we modify the Trojan by decreasing the delays of the trigger signals and run experiments using the best parameters for the attack previously found. The delays of the trigger signals are 2.17 ns and 2.227 ns, i.e., 10–15% lower than the smallest delay in Table 4. On the same FPGA instance as in Section VI-B, we activate 15 blocks of ROs (the range that causes leakage in Fig. 16), with an enable signal period of 19 clock cycles (Fig. 10) and a duty cycle of 31.58%. We run ten experiments, each comprising 15 runs with fixed and 15 with pseudorandom plaintexts, and count how many times the key leaks to the output. The results show higher leakage for the fixed (approx. 75 times per experiment, on average) than for the pseudorandom plaintext (eight times per experiment, on average). The key becomes the most occurring value after 53,248 encryptions, a result similar to the one reported in Section VI-A2. With this attack configuration, we sometimes observed FIFO resetting, but not in a manner that stops the exploit.

Repeating the experiment on another FPGA instance initially resulted in no significant leakage. However, with slight change of parameters, a significant number of leakage events were captured. The attacker size was 12 blocks (in the range that causes leakage in Fig. 16). The power wasters were enabled with a period of 16 clock cycles (Fig. 10) and a duty cycle of 31.25%. As a result, the key was found to leak approximately 88 times per fixed-plaintext experiment, on average, and 30 times per pseudorandom-plaintext experiment, on average. The key becomes the most occurring value after 20,480 encryptions.

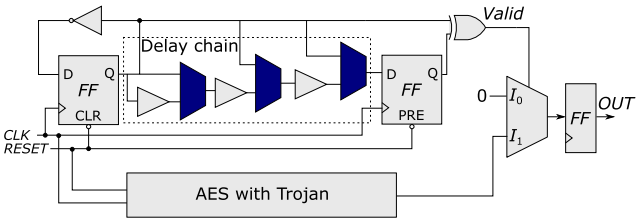


FIGURE 19. Our design of a protection against X-Attack. This example uses only three buffers in the delay chain. In normal operation, *Valid* is set and the AES output is routed to the *OUT* port. However, under the attack, the signal passing through the delay chain suffers from a timing fault; this results in clearing the *Valid* signal, thus forcing *OUT* to constant zero.

Finally, it is worth noting that because this setup is the most conservative due to fast trigger signals, the key sometimes may not be the most occurring value at the output (but it remains one of the most frequently occurring values). We repeat the experiments on five different FPGA instances and, on all of them, we successfully inject faults and induce leakage, proving that the risk associated with X-Attack is relevant across different devices from the same FPGA family. These experiments further show that using the insights obtained through the parameter sweeps and the analyses we performed or a similar offline evaluation that the adversary can carry out facilitates the attack.

VII. COUNTERMEASURES

With our understanding of X-Attack, we aim to design a lightweight and easy-to-implement defense. There are many potential countermeasures against X-Attack, some even requiring new FPGA designs, as we discuss in Section IX. However, our design targets protecting hardware running on currently available FPGA-based systems. Consequently, the main criteria for our countermeasure are the prevention of exploitable leakage, ease of deployment, and low usage of resources. First, we present below the initial design we proposed in our previous work [22]. Then, we discuss how we improve it for easier deployment in cloud environments.

A. INITIAL COUNTERMEASURE DESIGN

The main idea behind our countermeasure is introducing a combinational path that will fail if the voltage within the circuit drops enough to induce faults or leakage. Accordingly, the protection circuit leverages a delay chain (a sequence of buffers) that we ensure matches the delay of the longest path within the victim to protect. We show the countermeasure design in Fig. 19. We instantiate the delay chain between two registers, one of which takes as input the output of the delay chain, while the other's input is the inverse of the input to the delay chain. As a result, the two registers should generate opposite signals, assuming normal operating conditions. Since the *Valid* signal is the result of XORing the two outputs, it will be set, and the multiplexer's output will match the victim output.

When an adversary lowers the on-chip voltage, the increase in delay will affect the delay chain. Consequently, timing

faults should be observed in the output of the register whose input is supplied by the delay chain before or at the same time faults start occurring within the victim circuit. When this happens, the *Valid* signal becomes 0, and the output of the multiplexer receives a dummy value. The dummy value prevents the adversary from gaining information from the faulty output. Furthermore, the output can be connected to an interface to monitor such attacks and take action. The dummy value indicates to the interface the existence of a problem without requiring a validation of the correctness of the encryption through redundancy or by decrypting the ciphertext to compare the plaintexts. Since no data is going through the delay chain, the circuit is more likely to fault than the victim, where the paths taken to produce output signals during each encryption are data-dependent. Accordingly, if the delay of the countermeasure is calibrated well, the countermeasure should replace the output with the dummy value before any leakage occurs.

When implemented on the Intel DE1-SoC, the protection circuitry requires only 76 ALMs, i.e., only 2.84% of the number of ALMs used by the AES core. Moreover, we only add a multiplexer on the victim's path, while the rest of the protection hardware is separate. This separation minimizes the effect of the countermeasure on the maximum design clock frequency; in our experiments, the design clock frequency remains unchanged. It is also possible to add a flip-flop after the multiplexer to guarantee that transient effects will not change the final output, increasing the latency by only one clock cycle.

When testing the countermeasure, we first validate that it does not affect the victim when there is no attack or the attack is not strong enough to cause faults. We increase the delay of the trigger signals used in the AES by using equation 1 on SDC condition 00_{inv} to place them on the critical path and increase the likelihood of leakage. Then, we test several configurations where we manually calibrate the delay of the buffer chain in the range from 0.3 ns below to 1.5 ns above the AES critical path delay. Depending on the delay difference, we sometimes observe faulty ciphertexts. However, the key does not leak to the output in all cases. As expected, when the delay of the buffer chain is larger, we observe no faults as the countermeasure is activated earlier.

B. AUTOMATIC CALIBRATION AND CONTROLLABLE PARAMETERS

Our proposed countermeasure for X-Attack requires calibrating the buffer chain to match the critical path delay or the clock period. In our previous work, we manually calibrated the delay chain, which is feasible in an embedded setup [22]. However, when deploying designs to the cloud, it is desirable not to rebuild to test the calibration. Therefore, we augment our countermeasure with a self-calibrating functionality. The only thing the user would need to investigate and decide on is the length of the delay chain to use for calibration. The length should not be too short as to prevent the calibration but long enough to have a delay equivalent to the clock period. The length should also be limited to ensure reduced resource utilization overhead. Given the limitations on the reconfigurability of clocks within the cloud, we only calibrate to the clock period, which makes the defense very sensitive to increases in delay. Avoiding dummy outputs when the delay increases but not enough to affect the victim to be protected requires calibrating to the unknown critical path. It is difficult to compare to the critical path in practice because the delays can depend on the data.

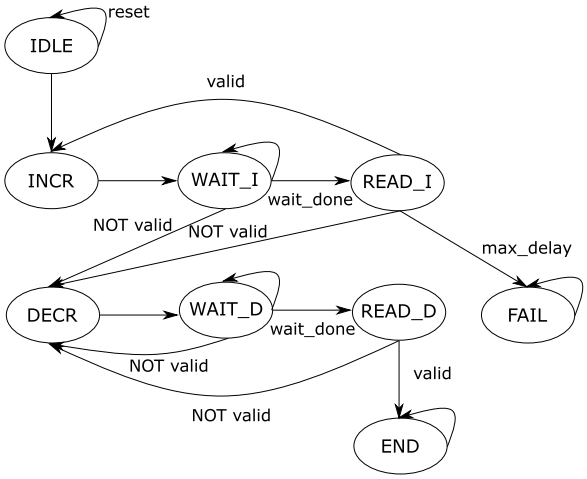


FIGURE 20. Calibration FSM for the countermeasure.

The main changes to introduce self-calibration to the design are as follows. First, we connect MUXes to the delay chain such that the input of each buffer comes from the output of its corresponding MUX. The MUX's inputs come from the previous buffer and the first register. By changing the select signals of the multiplexers, we change the delay of the chain. We show the modified buffer chain in Fig. 19 (the dark blue MUXes represent the modifications to the delay chain). Quartus combines the buffer and the MUX into one LUT so that the multiplexers do not introduce additional delays. The second change is the introduction of a calibration FSM to test the delay chain and calibrate the delay chain to the clock period. We show the FSM in Fig. 20.

The FSM starts in the IDLE state, where it remains as long as the reset is asserted. If the reset is asserted while the FSM is in any other state, the FSM goes back to the IDLE state. Within the IDLE state, the select signals of the multiplexers are all 0 so that the output of the first register is directly connected to the input of the second register. Therefore, once the FSM no longer receives an asserted reset signal, the FSM goes to INCR state, which increments the number of MUXes used by changing the last select to 1. Every time the FSM goes to the INCR state, one more select line changes to 1. After each increment, the FSM goes to the WAIT_I state, where it waits for a specified number of clock cycles for the change in the length of the delay chain to take effect. While waiting, if the logic detects that the *Valid* signal changes to 0, the state changes to DECR. Otherwise, when the waiting

period is done, the FSM goes to the READ_I state, where it checks the length of the delay chain and reads the *Valid* signal. If the delay chain uses the maximum number of buffers and the *Valid* signal remains 1, the calibration has failed. The FSM goes to FAIL state where it remains. If the *Valid* signal remains 1, and the countermeasure has not used the maximum number of buffers, the FSM goes back to the INCR state. The reason for using the WAIT_I state is to guarantee that if the increased delay causes metastability, the FSM can still capture the *Valid* signal changing to 0 and change the chain length accordingly. In the READ_I state, if the *Valid* signal is 0, the FSM moves to the DECR state to decrement the number of buffers by one to return to a valid state, where the delay is calibrated to the clock period. To guarantee that no issue has occurred, we also include the two states WAIT_D and READ_D to wait for the change in the multiplexers to take effect and to ensure that the final state of the multiplexers results in a *Valid* signal equal to 1. If that is the case, the calibration was successful, and the FSM goes to the END state, where it remains until reset. If for some reason, the *Valid* signal is still 0, the FSM goes back to the DECR state to remove an additional buffer. Once in the END state, the calibration value can be read and possibly overwritten if the calibration has rendered the countermeasure too sensitive.

We test the self-calibrating countermeasure in the cloud setup with 50 buffers in the delay chain and the SDC condition 11. The defense calibrates with about 10 buffers, depending on the FPGA instance assigned, as the delay of the chain, if it uses all 50 buffers, is 11.891 ns. We choose to instantiate more buffers than those required to match the clock period of 3.125 ns to ensure that the calibration will be successful. The countermeasure uses around 5% of the resources used by the AES (113.5 adaptive logic modules for the countermeasure and 2336 for the AES). The delays of the trigger signals t_0 and t_1 are 2.68 and 2.634 ns. The attack uses an enable period of eight clock cycles and a duty cycle of 62.5%.

Given that our countermeasure calibrates to the clock period, and not the critical path, when the ROs are all inactive, there is still a chance that the delay will momentarily increase and that the dummy value will appear at the output. However, since the design supports overwriting the calibration value, the user can slightly adjust the calibration to eliminate any false positives. Depending on the FPGA instance assigned by the CSP, the calibration can succeed without any false positives, or can need some small adjustments. We validate that, with the selected calibration value, when the ROs are inactive, the dummy value does not appear.

On the same FPGA instance as in Section VI-B, the countermeasure calibrates using eight buffers. Launching the attack with an increasing number of ROs, we observe that the countermeasure starts blocking the value of the ciphertext with three RO blocks, as shown in Table 7. The countermeasure activation rate surpasses 10% with six RO blocks. We also validate that up to 16 RO blocks, the key never leaks to the output, despite the trigger signals reaching the SDC condition at 12 RO blocks or more. For large attacker sizes, faults still appear at the output but at a reduced rate compared to the case without a countermeasure. The faults are most likely due to faults that occurred in earlier stages of the pipeline, and propagated to the output at a time when the ROs and the countermeasure were inactive.

TABLE 7. Proportion of dummy values at the output for a range of attacker sizes.

RO Blocks	Percentage
0	0
1	0
2	0
3	0.01%
4	0.37%
5	6.2%
6	10.58%
7	12.35%
8	18.09%
9	26.9%
10	27.6%
11	27.85%
12	28.01%
13	29.27%
14	28.85%
15	29.22%
16	30.8%

We repeat the attack with the countermeasure active on another FPGA instance. The countermeasure calibrates with nine buffers. We report the results for this board in Table 8. The countermeasure activation rate surpasses 10% with four blocks of ROs. The higher reactivity is to be expected, given the extra buffer in the calibration. On this FPGA instance, the FIFOs reset with 16 blocks of ROs, which is why Table 8 contains only 15 entries.

VIII. DISCUSSION

This work demonstrates that, in the FPGA multitenancy context, combining stealthy hardware Trojans with remote undervolting can threaten the security of FPGA-based applications. We show how to deploy and control FPGA power wasters to remotely create power supply voltage fluctuations, with the goal of activating an SDC Trojan by faulting the trigger signals to a normally never-reached state. Once activated, the SDC Trojan can leak a secret—in our case, the encryption key of an AES—to the output of the victim module. We have analyzed the effect of the attacker and Trojan parameters on the exploit's success in an embedded and cloud setup. We have also proposed a lightweight self-calibrating countermeasure suitable for deployment in

TABLE 8. Proportion of dummy values at the output for a range of attacker sizes and a different FPGA instance.

RO Blocks	Percentage
0	0
1	0
2	0.09%
3	3.88%
4	10.73%
5	12.3%
6	14.83%
7	24.6%
8	29.9%
9	29.74%
10	31.84%
11	31.68%
12	33.62%
13	35.18%
14	37.7%
15	40.73%

cloud environments. In this section, we highlight interesting observations derived from our results and discuss the generality and limitations of X-Attack.

A. ANALYSIS OF X-ATTACK

The first notable result is the difference between the attack results in embedded and cloud settings. Many things differ between the two setups, from the delays of various design elements to the maximum clock frequency that can be used and the attacker parameters for a successful exploit. While the possibility of fault injection exists in both setups, an adversary would need to spend some calibration effort to port the attack from one platform to another.

In our experiments, we use two types of workloads as plaintexts, the pseudorandom and the fixed set of plaintexts (see Section VI-B). As a consequence, the results of various analyses also differ. This difference in the results highlights how the signal paths activated (and therefore, the combinational delays for one specific encryption), the values of the signals and how they change, and, accordingly, the likelihood of fault injection are data-dependent. We show that, once the voltage is lowered, a signal with a long delay, whose value is not constant throughout the clock cycles when the undervolting occurs, is more likely to fault than others. We predicted the faults most likely to be observed in the trigger signal pair and found that experiments confirm our predictions. The effects of the signal values and whether they switch between '0' and '1' are why, for pseudorandom plaintexts, we observed leakage with fewer attacker blocks than for fixed plaintexts. All this means that, even with remote undervolting, to increase the chances of a successful attack, the adversary will likely have to repeat the exploit several times. Repetitions allow trying with various workloads (either by controlling the input or by waiting for different input values from other sources to the victim) until the trigger signals fault to the desired SDC state and the secret leaks.

The chosen SDC condition and the delays of the trigger signals are also decisive factors for the exploit's success. When inserting the Trojan, the adversary should consider the best SDC condition (i.e., the one that is most susceptible to faults). However, we have seen that the effect of the SDC condition on the attack's success depends on the properties of the input dataset. Without any knowledge of the input, the chosen SDC condition may be suboptimal, rendering the attack more challenging and requiring more trials. Once the SDC condition is chosen, the adversary will likely want to select a pair of suitable trigger signals that are close in delay to the critical path of the victim module. The higher the clock frequency at which the victim is running (e.g., close to the maximum operating frequency), the more likely the attack is to succeed.

Finally, our work demonstrated successful fault injection on cloud platforms, using a relatively small fraction of available resources. Even after accounting for additional resources for changing the design of the power wasters to be stealthier (e.g., adding latches) or for creating larger voltage drop to fault designs operating at lower than the maximum frequency, the risk of fault injection would persist. Whether the faults would be exploitable and in what way would ultimately depend on the victim functionality and the goals of the adversary in charge of inserting the Trojan (e.g., neural networks, encryption circuits, etc.) [18], [19], [22].

B. GENERALITY OF X-ATTACK

We demonstrate X-Attack on two members of the Intel FPGA families: Arria 10, one of the state-of-the-art commercially available cloud FPGA devices, and Cyclone V, commonly used in embedded systems. Nevertheless, we believe X-Attack is a threat to other FPGAs because, as we will explain shortly, all aspects of the exploit can be generalized.

The underlying mechanism enabling the remote undervolting part of the exploit is the same across most (if not all) available FPGA boards. Gnad et al. were the first to demonstrate remote undervolting to reset an FPGA board, testing on AMD boards [23]. Later work showed that an adversary could also instantiate power-wasting circuits on Intel FPGAs [57]. Zhu et al. systematically analyzed the PDN of AMD FPGA boards to understand the mechanism behind side-channel and fault-injection exploits [28]. The results on the relationship between the PDN resonance frequency and the generated voltage drop were also demonstrated on other AMD boards for fault-injection exploits [42]. Our work carries out a similar analysis on Intel FPGAs. Therefore, the undervolting aspect of the exploit can be adapted to

various FPGA PDNs to ensure successful undervolting of the chip. Furthermore, the evaluation we present on a commercially available cloud FPGA, without access to the PDN parameters, indicates that the optimal attack parameters can be experimentally found for various boards.

An adversary in a realistic scenario would need to test the power-wasting circuits on the target platform to map out the limits of the parameters to use and their expected effects. To ensure the ability to test without detection, the adversary can test on a local instance of the FPGA (if available), or rent an instance as a single tenant. The adversary will also need to gradually increase the strength of the exploit to ensure that an accidental denial of service does not occur and trigger a warning to the CSP. Depending on the target platform and where it is deployed, the adversary may need to change the power wasters to avoid checks for combinational loops, or to leverage other logic resources within the programmable logic region. There will also be some differences depending on the specific FPGA instance. However, as shown in our results (in Section VI, and Appendix B), the variations across FPGAs do not change the observed trends regarding which parameters control the attack's success.

The second essential component of X-Attack is the Trojan. The multiplexers and registers (the Trojan components) are readily available as basic components of all commercially available FPGAs. Furthermore, given how the Trojan is designed, once inserted into the victim design, the synthesis tools should not optimize it away because, first, the trigger signals are freely switching, and, second, the combinational blocks in which the trigger signals originate and in which the Trojan is, are separated. Our work tests the Trojan insertion with two different FPGAs from Intel, and we validate that Quartus does not automatically remove the Trojan. Previous work [24] has also tested the Trojan on an AMD board (SASEBO-W board with a Xilinx XC6S150T FPGA), and reported that the Trojan was not optimized away. We have also validated that the Trojan remains for an AMD Zynq UltraScale+ board when synthesized with Vivado. Moreover, our results in Section VI-B highlight how the trigger condition of the Trojan can change and what factors govern the exploit's success. This means that an adversary targeting a different victim module than the AES circuit used here can still find and choose a pair of trigger signals with a specific SDC condition and adapt the Trojan accordingly.

An adversary aware of the existence of a module compromised by an SDC Trojan will aim to ensure colocation with the victim [58]. Colocation is possible for a malicious CSP, for a malicious tenant in a multitenant cloud environment, or through the deployment of a malicious application within an embedded system. The adversary can then launch X-Attack, and collect the victim output to test for leakage. In the case of an attack against AES, the adversary can collect the most occurring ciphertexts, and test them as the key to see if the plaintext can be recovered from the observed ciphertexts. Given that the two aspects of X-Attack can be generalized to other targets, X-Attack is a threat to various applications on multitenant FPGAs.

C. LIMITATIONS OF X-ATTACK

While X-Attack is a strong attack vector, it still has some limitations. The first concerns the Trojan and the process of implanting it. A malicious party must be involved in the supply chain of an IP that will be handling secrets to be able to hide the Trojan within the IP. Depending on the target victim circuit, that malicious party needs to carefully choose the trigger signals and the placement of the Trojan, to ensure it cannot be detected, while ensuring that the activation of the Trojan by faulting the circuit is not unlikely. Once the Trojan is successfully hidden within the IP which is then used by the victim, the attacker still faces many challenges. Namely, the adversary cannot guarantee that when the Trojan is included in the victim design, its triggers will be anywhere near the critical path. Specifically, if the Trojan paths have a much shorter delay as compared to other circuit components, increasing their delay to trigger secret leakage might become infeasible. As we have discussed in Section VI, increasing the strength of the power wasting circuits can lead to the board resetting.

The second challenge that the adversary faces is to ensure colocation with the victim that uses the Trojan. While challenging, this colocation is not impossible. In the most constrained scenario, which is cloud multitenancy, the adversary can instantiate their design within various FPGAs, and then use a side channel to determine the tenants sharing the FPGA fabric until the victim is found [58]. Furthermore, the adversary needs to be able to observe the victim's output to be able to distinguish the secret leaking. Accordingly, the adversary needs to be able to establish communication with and send requests to the victim, or establish a side channel to monitor the output. Finally, as the attacker has no knowledge of the secret prior to the attack, and potentially has no control on the input, ensuring the switching of the trigger signals can also be difficult. As we have shown in Section VI-B, the switching of the trigger signals and the distribution of their values relate to the success of X-Attack. Despite the strength of X-Attack, the adversary will still face the limits of factors outside their control that affect the likelihood of the attack succeeding.

IX. RELATED WORK

This section presents related work on hardware Trojans, remote fault injection, and countermeasures. We also highlight how our work contributes to the research on remote fault injection.

A. DON'T-CARE HARDWARE TROJANS

The widespread use of electronic chips and the manufacturing complexity have opened the door for new vulnerabilities. Hardware Trojans, especially, have emerged as a potential threat with the involvement of many parties in the design and fabrication of integrated circuits. Consequently, many

researchers have examined the various types of Trojans, their insertion, and their triggering [59]. The increased research on hardware Trojans and the improved detection methods have led to stealthier Trojan designs, specifically those which use don't-care conditions. For example, Fern et al. leverage external don't-care conditions for Trojan insertion [43]. The lack of design specification for these external don't-cares makes the Trojan hard to detect, as the user cannot check whether the behavior is as expected. The unspecified functionality can also exist in a finite state machine, which Nahiyan et al. combine with fault attacks to send the design into a Trojan state, where the design performs malicious activities instead of the specified functionality [60]. Krieg et al. showed how to trigger a Trojan using the discrepancy between the behavioral simulation and the hardware implementation of a don't-care state [61]. Prior to deployment on hardware, the simulation will show that the design with the Trojan is equivalent to the specification, making the malicious modification challenging to detect. Hu et al. used internal don't-care conditions as a Trojan trigger [24]. Finally, Hu et al. targeted obfuscated designs by focusing on incorrect obfuscation keys as unspecified functionality to insert the Trojan [62]. Works on hardware Trojans assume a strong threat model, where the adversary has physical or logical access to the design. The attacker activates the Trojan by sending a specific input sequence, manipulating the clock frequency, or injecting faults. Then, the attacker can probe the Trojan payload. Our work instead removes the requirement of physical and logical access. The only requirements for our exploit are cotenancy with the victim circuit and the ability to observe its output.

B. ATTACKS IN MULTITENANT FPGAS

With the adoption of FPGAs in the cloud, security researchers have examined potential security threats affecting remotely accessible FPGAs. Some researchers have questioned whether the CSP is a trusted entity in such a case, and offered solutions to encrypt the designs deployed by the users, while still ensuring that they are safe to deploy within the cloud [63]. However, given the multitenancy customary in cloud platforms, the possible malicious efforts of a cotenant on the same FPGA have been a focus of many research works. Gnad et al. were the first to use ROs to waste power to reset the board for a denial-of-service attack [23]. Krautter et al. leveraged the power wasters to inject faults suitable for DFA against a colocated AES module, while Mahmoud et al. showed how the voltage drop could bias a true random number generator [18], [22]. Both works have shown that careful control of the ROs transforms the attack from DoS to exploitable fault injection. Zhu et al. have modeled the PDN of FPGA boards, and their work demonstrates the importance of tuning the frequency of the enable signal, as the voltage drop is at its maximum when the frequency is close to the resonance frequency of the PDN [28]. Researchers have also demonstrated successful fault injection exploits leveraging other attack primitives (e.g., block random access memory, register- and latch-based ROs, and overclocked AES encryption rounds) [26], [38], [39], [40].

In addition to fault-injection exploits, researchers have examined side-channel leakage in multitenant FPGAs. Sensors implemented within the programmable fabric can sense power variations within the same chip or from another chip to gain side-channel information [64], [65]. The sensors can enable side-channel exploits in deployed commercial FPGAs as Glamočanin et al. show in their work targeting an Amazon EC2 F1 instance [5], [27]. An adversary can utilize the side-channel leakage to steal encryption keys and neural network inputs and structure [66], [67]. Furthermore, power variations are not the only channels for leakage, as crosstalk-coupling between neighboring long wires in an FPGA facilitates covert communication and side-channel exploits [68], [69].

Our work is a remote fault-injection exploit targeting multitenant FPGAs. However, X-Attack assumes the existence of a hardware Trojan within the victim circuit. We leverage the remote undervolting to activate a stealthy SDC Trojan and target a commercial cloud FPGA instance. Additionally, unlike DFA, we do not target a specific encryption round; if the fault affects the trigger signals of the Trojan, the secret leaks directly to the output of the target hardware module.

C. COUNTERMEASURES

Given the myriad security issues for integrated circuits in general and FPGAs in particular, researchers have devoted significant efforts to defense mechanisms. X-Attack combines hardware Trojans with FPGA-based undervolting. Researchers have examined both hardware Trojans and power wasters.

On the hardware Trojans side, many defense techniques require access to a golden design against which to compare or rely on applying various input patterns to test for abnormal behavior [70]. However, the reference design may only sometimes be available to compare against. Moreover, only an injected fault activates an SDC Trojan, which means that testing with various input values will not activate the Trojan. Dai and Yavuz have proposed a countermeasure focusing on the don't-care states in FSMs. Their detection approach works at the gate and register transfer levels and requires no golden design [71]. Given the stealthiness of SDC Trojans, researchers have looked into specific detection methods for them. Wu et al. propose a method to look for specious LUTs, or LUTs where certain entries cannot be covered due to SDC conditions [72]. Hu et al. show how integrating information flow tracking into high-level synthesis design flows can detect the design modifications necessary for an SDC Trojan [73].

On the remote undervolting side, the defense mechanisms either focus on the malicious circuit or the undervolting effects. It is also possible to focus on the architecture of

future FPGA devices. For example, Ahmed et al. proposed the design of an optimized LUT with input-to-output delays with decreased sensitivity to changes in the supply voltage. They also proposed using separate voltage islands for LUTs and routing [74]. For the detection of malicious circuits, Krautter et al. [75] and La et al. [26] focused on searching for malicious circuits in the final design bitstream. They achieved it by building bitstream scanners, reconstructing the design netlist, and searching for specific patterns suggesting potentially malicious circuits. The developed scanners target the Lattice iCE40 [75] and the AMD-Xilinx Ultrascale+ [26].

On the other hand, many researchers focus on detecting the effects of power wasters. For instance, the detection can rely on instantiating multiple instances of security-critical functions and checking that their outputs match. Given the area overhead of redundancy countermeasures, researchers have also proposed sensing-based countermeasures. These include glitch detectors [76], aging sensors [77], [78], shadow registers [79], and razor latches [80]. Researchers have also proposed detectors for voltage drops, which can suppress clock edges to avoid fault injection [81]. The cloud service provider can even deploy voltage sensors within the FPGA in a distributed fashion to locate the malicious tenant [57], [82]. The detection of the lower voltage can be combined with a mechanism to disable the offending circuit to limit the damage it can do, as proposed by Nassar et al. [83]. The main differences between our design and previous proposals are its independence from the victim and self-calibration capabilities. Our defense does not include a correction mechanism but simply disconnects the faulty output and sends an alarm using the dummy output.

X. CONCLUSION

The integration of FPGAs in the cloud has led to multitenancy proposals to offer efficient resource utilization. However, the low-level programmability of FPGAs introduces a variety of electrical-level security threats. A thorough investigation of the vulnerabilities is necessary to propose a better design for FPGAs to integrate safely into the multitenancy model of cloud environments. In this work, we have investigated the undervolting resulting from power wasters within an FPGA on embedded and cloud FPGAs. We have also shown how an adversary can employ the lowered voltage to activate a stealthy SDC Trojan within a cryptographic core. Our analysis examined attacker, Trojan, and deployment parameters to determine their effect on the probability of a successful exploit. Finally, we have designed a self-calibrating countermeasure to protect circuits from the effects of undervolting. Our work highlights the need for research on secure multitenant designs for cloud FPGAs, before allowing multitenancy in the cloud. Future work can focus on testing X-Attack with other victim circuits or on future devices. Additional work can also examine the power isolation of tenants and the detection of stealthy Trojans and stealthy power waster designs.

APPENDIX A
ENCRYPTION KEYS AND PLAINTEXTS

To facilitate the reproducibility of the results presented in this work, we list below the AES key and the hardcoded (fixed) plaintexts used in the embedded and the cloud setup.

A. EMBEDDED SETUP

```
Key:
0x85458A2BB4A9AAFD C5620273D3CF034A

Plaintexts:
0x0000000000000000 0000000000000051
0xDA705E312B8D9705 7E94B4A810D531EF

Ciphertexts:
0xF745C51C0911D0F4 D1C2C6B69A894F42
0xE3657A2422FA7811 86C4045A7202EABA
```

B. CLOUD SETUP

```
Key:
0x7E151628AED2A6AB F7158809CF4F3C0D

Plaintexts:
0xDA705E312B8D9705 7E94B4A810D531EF
0xAB93743290CD9432 7432427FE4580327
0xFFFFFFFFFFFFFFFF FFFFFFFFFFFFFF29
0x7142127714212771 4212771421270123
0x0000000000000000 0000000000000025
0x1234560000000000 0000000000000026
0xABCD000000000000 0000000000000027
0x0000000000076543 0000000000000028
0xFFFFFFFFFFFFFFF0 FFFFFFFFFFFFFF29
0x0000000000000000 0000000000000038
0x0000000000000000 0000000000000050
0x0000000000000000 0000000000000049
0x0000000000000000 0000000000000046
0x0000000000000000 0000000000000047
0x0000000000000000 0000000000000045
0x0000000000000000 0000000000000051

Ciphertexts:
0x2F4032BB10F20482 FFE3985BBC186B0D
0x2F29AE6E636C7D5C 0E8CCB3E1AE085BE
0xF4A84F93AB21D9C5 16D252967A0B9896
0x7FFFEB39C2E8899E AFF8899048A9A60D
0x2F2A29D754DF610D 7E3BEDA124E2B568
0x6046E22933797889 92159155AC270E95
0x3FF240968A572F9D 87B8ED66C87C1BB6
0xE5E884F43925E9E9 4A4E2E0F0379ECA5
0x07D7454AD710758C 57797AEDB02278A8
0xD6529F90F7A205F4 A507B8FB192AAAD9
0x8F889A12C593309D EA30C1AAD938C314
0x50177771DD1D942F 91DFBD130E0B99C0
0x1E3AEC3F75FDC0CC 1AFB8BB948E53F91
```

```
0x68BA6ED9D6E9068D DC4ACD5CFB2AF1D6
0xC34BFB11F6597820 2262E5166FF652A6
0xC40AD3CBF7E4FEB5 CEB586C4257B1255
```

APPENDIX B
RESULTS ON OTHER CLOUD FPGA INSTANCES

When a user requests a cloud FPGA instance in a specific geographical region, they get assigned one of the available instances, i.e., they cannot choose upfront the preferred instance. Similarly, when launching our experiments, we would get access to different FPGAs. Given that every instance has an identification number, we could choose to either proceed with the experiments or request another instance at a later point in time. In Section VI, we show results of the experiments performed on the same FPGA instance.[3]

To assess the generality of X-Attack, we repeated the experiments on different FPGA instances. In this appendix, we show a sample of the results obtained using a second Intel Arria 10 cloud FPGA instance provided by the same CSP. We chose not to repeat the sweep of the enable signal frequency, because changing the frequency poses the highest risk of resetting the board. The potentially different sensitivity of other boards to the undervolting can lead to a DoS, which we wanted to avoid. Hence, we used the results in Fig. 10 as general guidelines for choosing the enable signal parameters irrespective of the allocated FPGA instance. In what follows, we discuss the results of the additional experiments.

1) AVERAGE LEAKAGE OCCURRENCE

Fig. 21 shows the average number of AES key leakage occurrences for each tested SDC condition, multiple attacker sizes, and pseudorandom plaintexts (similar to Fig. 17). The attacker size varies between 12 and 15 blocks. We do not show the results for 16 blocks because, on this instance, the FIFOs were likely to reset when the attacker size exceeded 15 blocks. In comparison, on the FPGA instance used in Section VI-B, 16 blocks would not cause FIFO reset. From our experience, minor differences in board-level sensitivity to undervolting are to be expected. With logic delays increasing and metastability, some variability in the results is inevitable which is why we report the average across a large number of experiments. Minor differences aside, general trends are consistent: First, the SDC conditions resulting in the highest, respectively lowest, average number of leakage occurrences are the same between the two FPGA instances. Second, the leakage occurs more often as the attacker size increases, consistent with Fig. 16. Last but not least, as the figure does not show data for the attacker of 16 blocks, it is worth mentioning that we did observe leakage for all SDC conditions, including 00.

[3]An exception is the experiment with varying distance between the adversary and the victim (Section VI-C), for which we were unable to get access to the desired instance.

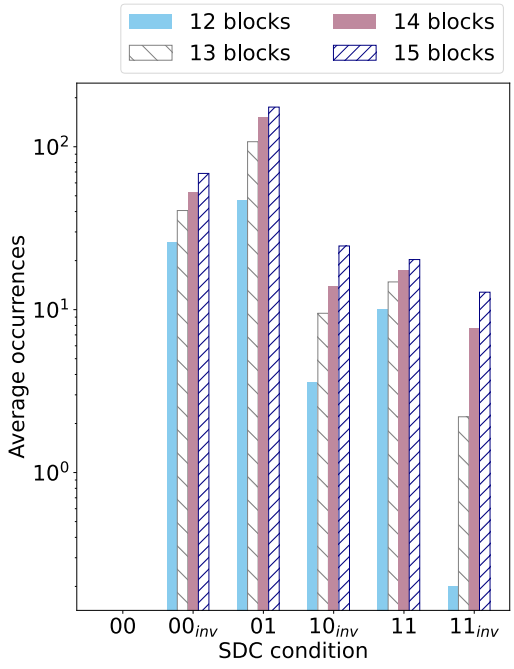


FIGURE 21. Average leakage occurrences (log-scale) for each tested SDC condition, multiple attacker sizes, and pseudorandom plaintexts for the second FPGA instance.

TABLE 9. Summary of the average number of faults observed for the 11 SDC condition with various attacker sizes for the fixed plaintexts case. The numbers are averaged over ten experiments each containing 15 runs. One run consists of 4,096 encryptions. The results are for a different FPGA than the one used in Table 1 In bold, the faults leading to leakage. In italic, the values corresponding to double bit flips.

Blocks		00	01	10	11
12	00	-	0.59	0	**0**
	01	4.52	-	*0*	**0**
	10	0.01	*0*	-	**0.01**
13	00	-	26.14	0	**0**
	01	38.35	-	*0*	**0**
	10	3.41	*0.37*	-	**3.11**
14	00	-	90.93	0	**0**
	01	75.85	-	*0*	**0**
	10	12.07	*11.43*	-	**20.29**
15	00	-	157.77	0.37	**0**
	01	137.45	-	*0*	**0**
	10	0.71	*34.95*	-	**20.59**

2) AVERAGE FAULT OCCURRENCE

Table 9, similarly to Table 1 shows the average number of bit flips occurring in the trigger signal pair for the SDC condition 11, with varying attacker sizes. The two tables show the same trend of the faults leading to leakage increasing with the attacker size. They both also show that only one value of the trigger signals faults to the SDC condition and results in leakage.

Table 10, similarly to Table 6 in Section VI-B, shows the bit flips occurring in the trigger signal pair, for all tested SDC

TABLE 10. Average number of occurrences of the faults in the trigger signals for all tested SDC conditions over ten experiments each containing 15 runs with fixed plaintexts. The attacker has 15 blocks of ROs. Experiments run on a different cloud FPGA than the one used for Table 6. The values in bold denote the number of occurrences of the bit flips that lead to leakage. The values in italic indicate that the corresponding pair of trigger signals (second column) does not occur, meaning that none of the transitions in that row can happen.

SDC		00	01	10	11
00	01	**0**	-	0	47.34
	10	**0**	*0*	-	*0*
	11	**0**	2.9	0	-
00_{inv}	01	**0.01**	-	15.29	60.07
	10	**0**	0	-	4.91
	11	**0**	0	20.42	-
01	00	-	**0**	0	0
	10	0	**0**	-	0
	11	*0*	**0**	*0*	-
10_{inv}	00	-	*0*	**0**	*0*
	01	21.53	-	**0**	230.33
	11	2.35	164.34	**0**	-
11	00	-	157.77	0.37	**0**
	01	137.45	-	0	**0**
	10	0.71	34.95	-	**20.59**
11_{inv}	00	-	0	0.29	**0**
	01	*0*	-	*0*	**0**
	10	0	0	-	**0**

conditions. As a reminder, only 15 RO blocks are activated on this second cloud FPGA instance and the reported numbers are likely to differ. Despite of that, the general trends these two tables show are the same: the expected faults indeed happen, the unexpected ones are not observed. One minor difference is that, on the second instance, we detected leakage with SDC condition 00_{inv}, albeit very rarely.

3) ATTACK DURATION REQUIREMENTS

Next, we analyze the number of encryptions (i.e., attack duration) required for the secret key to become the most occurring value at the victim's output. We run 10 experiments with 61,440 encryptions each (same setup as in Section VI-A2). In every experiment, we look at the N_{CT} obtained ciphertexts ($0 \leq N_{CT} \leq 61{,}440$) to count the number of occurrences of the secret key. If the key was the most frequently encountered value among the analyzed ciphertexts, we consider that attack duration sufficient for a successful attack. If the attack duration is confirmed sufficient in all 10 experiments, the probability (attack success in Fig. 22) is the highest value equal to one. Comparing Figs. 15 and 22, we see that on the second FPGA instance, for an attack to succeed across all 10 experiments (100% accuracy) more encryptions were required. For example, for 15 RO blocks, approx. 35k encryptions were needed on the second FPGA instance, compared to approx. 20k encryptions on the first FPGA instance. However, for 9 out of 10 attacks to succeed (90% accuracy), approx. 20k encryptions were sufficient on both instances. We find such differences both expected and

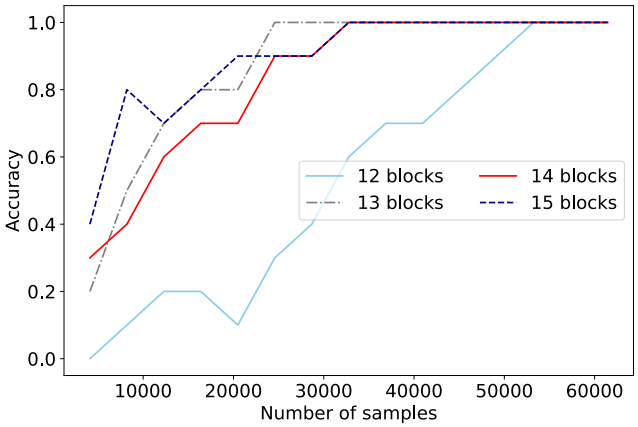


FIGURE 22. The probability that the key is the most occurring output value in the function of the number of obtained ciphertexts (i.e., output samples). The AES encrypts a sequence of pseudorandom plaintexts, while the number of RO blocks varies from 12 to 15. The horizontal axis corresponds to the number of analyzed ciphertexts. We refer to this probability as the accuracy of the prediction that the key will be the most occurring output value for a given number of output samples (i.e., attack duration).

acceptable. Finally, 13 blocks appear more effective than 14 or 15. A similar effect was observed on the first FPGA instance and explained in Section VI-A2.

4) CONCLUDING REMARKS

In conclusion, given that the fault injection can be affected by the process variations, aging, temperature, and possibly other effects the adversary has no control over, some differences across FPGA instances are expected. We observed and reported them. Importantly, we found that the general trends are consistent, confirming the portability of the attack across the target FPGA instances. In practice, these variations may aid the exploit as much as they can make it more challenging (see Section VIII-C).

REFERENCES

[1] *Cloud Computing Services—Amazon Web Services*. Amazon Web Services. Accessed: Jul. 10, 2023. [Online]. Available: https://aws.amazon.com
[2] A. Shawahna, S. M. Sait, and A. El-Maleh, "FPGA-based accelerators of deep learning networks for learning and classification: A review," *IEEE Access*, vol. 7, pp. 7823–7859, 2019.
[3] J. Hoozemans, J. Peltenburg, F. Nonnemacher, A. Hadnagy, Z. Al-Ars, and H. P. Hofstee, "FPGA acceleration for big data analytics: Challenges and opportunities," *IEEE Circuits Syst. Mag.*, vol. 21, no. 2, pp. 30–47, 2nd Quart., 2021.
[4] H. Li, Y. Tang, Z. Que, and J. Zhang, "FPGA accelerated post-quantum cryptography," *IEEE Trans. Nanotechnol.*, vol. 21, pp. 685–691, 2022.
[5] *FPGA-Based Amazon EC2 F1 Computing Instances*. Amazon Web Services. Accessed: Jul. 10, 2023. [Online]. Available: https://aws.amazon.com/ec2/instance-types/f1/
[6] (May 2022). *Compute Optimized Type Family With FPGA*. Alibaba Cloud. [Online]. Available: https://www.alibabacloud.com/help/en/elastic-compute-service/latest/compute-optimized-type-family-with-fpga
[7] C. Bobda, J. M. Mbongue, P. Chow, M. Ewais, N. Tarafdar, J. C. Vega, K. Eguro, D. Koch, S. Handagala, M. Leeser, M. Herbordt, H. Shahzad, P. Hofste, B. Ringlein, J. Szefer, A. Sanaullah, and R. Tessier, "The future of FPGA acceleration in datacenters and the cloud," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 15, no. 3, pp. 1–42, Sep. 2022.
[8] *Baidu Cloud Compute (BCC)*. Baidu AI Cloud. Accessed: Jul. 10, 2023. [Online]. Available: https://intl.cloud.baidu.com/product/bcc.html

[9] J. M. Mbongue, D. T. Kwadjo, A. Shuping, and C. Bobda, "Deploying multi-tenant FPGAs within Linux-based cloud infrastructure," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 15, no. 2, pp. 1–31, Dec. 2021.

[10] E. Chung, J. Fowers, K. Ovtcharov, M. Papamichael, A. Caulfield, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, and M. Abeydeera, "Serving DNNs in real time at datacenter scale with project brainwave," *IEEE Micro*, vol. 38, no. 2, pp. 8–20, Mar. 2018.

[11] *What is Virtualization?* Amazon Web Services. Accessed: Jul. 1, 2023. [Online]. Available: https://aws.amazon.com/what-is/virtualization/

[12] A. Vaishnav, K. D. Pham, and D. Koch, "A survey on FPGA virtualization," in *Proc. 28th Int. Conf. Field Program. Log. Appl. (FPL)*, Dublin, Ireland, Aug. 2018, pp. 131–138.

[13] R. Paccagnella, L. Luo, and C. W. Fletcher, "Lord of the ring(s): Side channel attacks on the CPU on-chip ring interconnect are practical," in *Proc. 30th USENIX Secur. Symp.*, Aug. 2021, pp. 645–662.

[14] Y. Wang, R. Paccagnella, E. T. He, H. Shacham, C. W. Fletcher, and D. Kohlbrenner, "Hertzbleed: Turning power side-channel attacks into remote timing attacks on x86," in *Proc. 31st USENIX Secur. Symp.*, Boston, MA, USA, Aug. 2022, pp. 679–697.

[15] X. Lou, T. Zhang, J. Jiang, and Y. Zhang, "A survey of microarchitectural side-channel vulnerabilities, attacks, and defenses in cryptography," *ACM Comput. Surv.*, vol. 54, no. 6, pp. 1–37, Jul. 2021.

[16] D. G. Mahmoud, V. Lenders, and M. Stojilović, "Electrical-level attacks on CPUs, FPGAs, and GPUs: Survey and implications in the heterogeneous era," *ACM Comput. Surv.*, vol. 55, no. 3, pp. 1–40, Feb. 2022.

[17] I. Giechaskiel, S. Tian, and J. Szefer, "Cross-VM covert- and side-channel attacks in cloud FPGAs," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 16, no. 1, pp. 1–29, Dec. 2022.

[18] J. Krautter, D. R. E. Gnad, and M. B. Tahoori, "FPGAhammer: Remote voltage fault attacks on shared FPGAs, suitable for DFA on AES," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, pp. 44–68, Aug. 2018.

[19] A. S. Rakin, Y. Luo, X. Xu, and D. Fan, "Deep-Dup: An adversarial weight duplication attack framework to crush deep neural network in multi-tenant FPGA," in *Proc. 30th USENIX Secur. Symp.*, Aug. 2021, pp. 1919–1936.

[20] S. M. Trimberger and J. J. Moore, "FPGA security: Motivations, features, and applications," *Proc. IEEE*, vol. 102, no. 8, pp. 1248–1265, Aug. 2014.

[21] F. Turan and I. Verbauwhede, "Trust in FPGA-accelerated cloud computing," *ACM Comput. Surveys*, vol. 53, no. 6, pp. 1–28, Dec. 2020.

[22] D. G. Mahmoud, W. Hu, and M. Stojilovic, "X-attack: Remote activation of satisfiability don't-care hardware trojans on shared FPGAs," in *Proc. 30th Int. Conf. Field-Programmable Log. Appl. (FPL)*, Aug. 2020, pp. 185–192.

[23] D. R. E. Gnad, F. Oboril, and M. B. Tahoori, "Voltage drop-based fault attacks on FPGAs using valid bitstreams," in *Proc. 27th Int. Conf. Field Program. Log. Appl. (FPL)*, Ghent, Belgium, Sep. 2017, pp. 1–7.

[24] W. Hu, L. Zhang, A. Ardeshiricham, J. Blackstone, B. Hou, Y. Tai, and R. Kastner, "Why you should care about don't cares: Exploiting internal don't care conditions for hardware trojans," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Irvine, CA, USA, Nov. 2017, pp. 707–713.

[25] O. Glamocanin, D. G. Mahmoud, F. Regazzoni, and M. Stojilovic, "Shared FPGAs and the holy grail: Protections against side-channel and fault attacks," in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, Grenoble, France, Feb. 2021, pp. 1645–1650.

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

[27] O. Glamocanin, L. Coulon, F. Regazzoni, and M. Stojilovic, "Are cloud FPGAs really vulnerable to power analysis attacks?" in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Grenoble, France, Mar. 2020, pp. 1007–1010.

[28] H. Zhu, X. Guo, Y. Jin, and X. Zhang, "PowerScout: A security-oriented power delivery network modeling framework for cross-domain side-channel analysis," in *Proc. Asian Hardw. Oriented Secur. Trust Symp. (AsianHOST)*, Dec. 2020, pp. 1–6.

[29] (Oct. 2020). *Intel Programmable Acceleration Card (PAC) With Intel Arria 10 GX FPGA Data Sheet*. Intel. [Online]. Available: https://www.intel.com/content/www/us/en/docs/programmable/683226/current/introduction-rush-creek.html

[30] (2020). *Zynq UltraScale+ Device Technical Reference Manual*. Xilinx. [Online]. Available: https://docs.xilinx.com/v/u/en-US/ug1085-zynq-ultrascale-trm

[31] M. Tehranipoor and F. Koushanfar, "A survey of hardware Trojan taxonomy and detection," *IEEE Des. Test Comput.*, vol. 27, no. 1, pp. 10–25, Jan. 2010.

[32] M. Tunstall, D. Mukhopadhyay, and S. Ali, "Differential fault analysis of the advanced encryption standard using a single fault," in *Proc. Inf. Secur. Theory Pract. Secur. Privacy Mobile Devices Wireless Commun.*, Heraklion, Greece, Jun. 2011, pp. 224–233.

[33] P. Dusart, G. Letourneux, and O. Vivolo, "Differential fault analysis on AES," in *Proc. Appl. Cryptography Netw. Secur.*, Kunming, China, Oct. 2003, pp. 293–306.

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

[35] T. La, K. D. Pham, J. Powell, and D. Koch, "Denial-of-service on FPGA-based cloud infrastructures—Attack and defense," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2021, no. 3, pp. 441–464, Jul. 2021.

[36] G. K. Yeap, *Practical Low Power Digital VLSI Design*. Berlin, Germany: Springer, Dec. 2012.

[37] O. Glamoćanin, A. Kostic, S. Kostic, and M. Stojilovic, "Active wire fences for multitenant FPGAs," in *Proc. 26th Int. Symp. Design Diag. Electron. Circuits Syst. (DDECS)*, May 2023, pp. 13–20.

[38] G. Provelengios, D. Holcomb, and R. Tessier, "Power wasting circuits for cloud FPGA attacks," in *Proc. 30th Int. Conf. Field-Programmable Log. Appl. (FPL)*, Gothenburg, Sweden, Aug. 2020, pp. 231–235.

[39] K. Matas, T. M. La, K. D. Pham, and D. Koch, "Power-hammering through glitch amplification—Attacks and mitigation," in *Proc. IEEE 28th Annu. Int. Symp. Field-Programmable Custom Comput. Mach. (FCCM)*, Fayetteville, AR, USA, May 2020, pp. 65–69.

[40] 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 *Proc. Workshop Fault Diagnosis Tolerance Cryptography (FDTC)*, Atlanta, GA, USA, Aug. 2019, pp. 48–55.

[41] D. Mahmoud and M. Stojilovic, "Timing violation induced faults in multi-tenant FPGAs," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Florence, Italy, Mar. 2019, pp. 1745–1750.

[42] D. G. Mahmoud, S. Hussein, V. Lenders, and M. Stojilovic, "FPGA-to-CPU undervolting attacks," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2022, pp. 999–1004.

[43] N. Fern, S. Kulkarni, and K. T. Cheng, "Hardware trojans hidden in RTL don't cares—Automated insertion and prevention methodologies," in *Proc. IEEE Int. Test Conf. (ITC)*, Anaheim, CA, USA, Oct. 2015, pp. 1–8.

[44] A. Tang, S. Sethumadhavan, and S. Stolfo, "CLKSCREW: Exposing the perils of security-oblivious energy management," in *Proc. 26th Usenix Secur. Symp.*, Vancouver, BC, Aug. 2017, pp. 1057–1074.

[45] S. Yazdanshenas and V. Betz, "The costs of confidentiality in virtualized FPGAs," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 10, pp. 2272–2283, Oct. 2019.

[46] C. Wolf. *Yosys Open SYnthesis Suite*. Accessed: May 10, 2023. [Online]. Available: www.clifford.at/yosys/

[47] (2020). *Development and Education Boards*. Intel. [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/technical/fpga-academic-boards.html

[48] H. Hsing. *Tiny AES*. Accessed: Jul. 10, 2023. [Online]. Available: https://opencores.org/projects/tiny_aes

[49] P. Alfke. (Jul. 1996). *Efficient Shift Registers, LFSR Counters, and Long Pseudo-Random Sequence Generators*. [Online]. Available: https://docs.xilinx.com/v/u/en-US/xapp052

[50] (Oct. 2018). *Deep Dive Into Alibaba Cloud F3 FPGA as a Service Instances*. [Online]. Available: https://www.alibabacloud.com/blog/deep-dive-into-alibaba-cloud-f3-fpga-as-a-service-instances_594057

[51] (Jun. 2021). *AWS Announces a New Shell for F1 Instances With Increased FPGA Resources and Data Transfer Speeds*. [Online]. Available: https://aws.amazon.com/about-aws/whats-new/2021/06/aws-announces-a-new-shell-for-f1-instances-with-increased-fpga-resources-and-data-transfer-speeds/

[52] *Open Programmable Acceleration Engine—Documentation*. Intel. Accessed: Jul. 15, 2023. [Online]. Available: https://opae.github.io/

[53] (Jul. 2020). *Accelerator Functional Unit Developer's Guide for Intel FPGA Programmable Acceleration Card*. [Online]. Available: https://www.intel.com/content/www/us/en/docs/programmable/683129/1-2-and-2-0-1/about-this-document.html

[54] (Apr. 2019). *Intel Acceleration Stack for Intel Xeon CPU With FPGAs Core Cache Interface (CCI-P) Reference Manual*. Intel. [Online]. Available: https://www.intel.com/content/www/us/en/docs/programmable/683193/current/about-this-document.html
[55] *OPAE SDK Source Code Repository*. Open FPGA Stack. Accessed: Jul. 15, 2023. [Online]. Available: https://github.com/OFS/opae-sdk/tree/master
[56] C. Drewes, O. Weng, K. Ryan, B. Hunter, C. McCarty, R. Kastner, and D. Richmond, "Turn on, tune in, listen up: Maximizing side-channel recovery in time-to-digital converters," in *Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays*, Monterey, CA, USA, Feb. 2023, pp. 22–111.
[57] G. Provelengios, D. Holcomb, and R. Tessier, "Characterizing power distribution attacks in multi-user FPGA environments," in *Proc. 29th Int. Conf. Field Program. Log. Appl. (FPL)*, Barcelona, Spain, Sep. 2019, pp. 194–201.
[58] M. Gobulukoglu, C. Drewes, W. Hunter, R. Kastner, and D. Richmond, "Classifying computations on multi-tenant FPGAs," in *Proc. 58th ACM/IEEE Design Autom. Conf. (DAC)*, San Francisco, CA, USA, Dec. 2021, pp. 1261–1266.
[59] K. Xiao, D. Forte, Y. Jin, R. Karri, S. Bhunia, and M. Tehranipoor, "Hardware trojans: Lessons learned after one decade of research," *ACM Trans. Design Autom. Electron. Syst.*, vol. 22, no. 1, pp. 1–23, May 2016.
[60] A. Nahiyan, K. Xiao, K. Yang, Y. Jin, D. Forte, and M. Tehranipoor, "AVFSM: A framework for identifying and mitigating vulnerabilities in FSMs," in *Proc. 53rd ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Austin, TX, USA, Jun. 2016, pp. 1–6.
[61] C. Krieg, C. Wolf, A. Jantsch, and T. Zseby, "Toggle MUX: How X-optimism can lead to malicious hardware," in *Proc. 54th ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Austin, TX, USA, Jun. 2017, pp. 1–6.
[62] W. Hu, Y. Ma, X. Wang, and X. Wang, "Leveraging unspecified functionality in obfuscated hardware for trojan and fault attacks," in *Proc. Asian Hardw. Oriented Secur. Trust Symp. (AsianHOST)*, China, Dec. 2019, pp. 1–6.
[63] S. Zeitouni, J. Vliegen, T. Frassetto, D. Koch, A.-R. Sadeghi, and N. Mentens, "Trusted configuration in cloud FPGAs," in *Proc. IEEE 29th Annu. Int. Symp. Field-Programmable Custom Comput. Mach. (FCCM)*, May 2021, pp. 233–241.
[64] F. Schellenberg, D. R. E. Gnad, A. Moradi, and M. B. Tahoori, "An inside job: Remote power analysis attacks on FPGAs," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Dresden, Germany, Mar. 2018, pp. 1111–1116.
[65] M. Zhao and G. E. Suh, "FPGA-based remote power side-channel attacks," in *Proc. IEEE Symp. Secur. Privacy (SP)*, San Francisco, CA, USA, May 2018, pp. 229–244.
[66] S. Moini, S. Tian, D. Holcomb, J. Szefer, and R. Tessier, "Power side-channel attacks on BNN accelerators in remote FPGAs," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 11, no. 2, pp. 357–370, Jun. 2021.
[67] Y. Zhang, R. Yasaei, H. Chen, Z. Li, and M. A. A. Faruque, "Stealing neural network structure through remote FPGA side-channel analysis," *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 4377–4388, 2021.
[68] C. Ramesh, S. B. Patil, S. N. Dhanuskodi, G. Provelengios, S. Pillement, D. Holcomb, and R. Tessier, "FPGA side channel attacks without physical access," in *Proc. IEEE 26th Annu. Int. Symp. Field-Programmable Custom Comput. Mach. (FCCM)*, Boulder, CO, USA, Apr. 2018, pp. 45–52.
[69] I. Giechaskiel, K. B. Rasmussen, and K. Eguro, "Leaky wires: Information leakage and covert communication between FPGA long wires," in *Proc. 13th ACM ASIA Conf. Inf., Comput. Commun. Secur. (ASIACCS)*, Incheon, Republic of Korea, Jun. 2018, pp. 15–27.
[70] S. Bhasin and F. Regazzoni, "A survey on hardware trojan detection techniques," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2015, pp. 2021–2024.
[71] R. Dai and T. Yavuz, "A symbolic approach to detecting hardware Trojans triggered by don't care transitions," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 28, no. 2, pp. 1–31, Dec. 2022.
[72] L. Wu, X. Li, J. Zhu, J. Zheng, and W. Hu, "Identifying specious LUTs for satisfiability don't care trojan detection," in *Proc. IEEE 34th Int. System-Chip Conf. (SOCC)*, Las Vegas, NV, USA, Sep. 2021, pp. 170–175.
[73] W. Hu, A. Ardeshiricham, L. Wu, and R. Kastner, *Integrating Information Flow Tracking Into High-Level Synthesis Design Flow*. Berlin, Germany: Springer, 2022, pp. 365–387.
[74] I. Ahmed, L. L. Shen, and V. Betz, "Optimizing FPGA logic circuitry for variable voltage supplies," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 4, pp. 890–903, Apr. 2020.
[75] J. Krautter, D. R. E. Gnad, and M. B. Tahoori, "Mitigating electrical-level attacks towards secure multi-tenant FPGAs in the cloud," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 12, no. 3, pp. 1–26, Aug. 2019.
[76] L. Zussa, A. Dehbaoui, K. Tobich, J.-M. Dutertre, P. Maurine, L. Guillaume-Sage, J. Clediere, and A. Tria, "Efficiency of a glitch detector against electromagnetic fault injection," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Dresden, Germany, Mar. 2014, pp. 1–6.
[77] Z. Ghaderi, M. Ebrahimi, Z. Navabi, E. Bozorgzadeh, and N. Bagherzadeh, "SENSIBle: A highly scalable SENsor DeSIgn for path-based age monitoring in FPGAs," *IEEE Trans. Comput.*, vol. 66, no. 5, pp. 919–926, May 2017.
[78] A. Amouri and M. Tahoori, "A low-cost sensor for aging and late transitions detection in modern FPGAs," in *Proc. 21st Int. Conf. Field Program. Log. Appl.*, Chania, Greece, Sep. 2011, pp. 329–335.
[79] E. Stott, J. M. Levine, P. Y. K. Cheung, and N. Kapre, "Timing fault detection in FPGA-based circuits," in *Proc. IEEE 22nd Annu. Int. Symp. Field-Programmable Custom Comput. Mach.*, Boston, MA, USA, May 2014, pp. 96–99.
[80] D. Ernst, N. Sung Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, "Razor: A low-power pipeline based on circuit-level timing speculation," in *Proc. 22nd Digit. Avionics Syst. Conf.*, San Diego, CA, USA, 2003, pp. 7–18.
[81] L. L. Shen, I. Ahmed, and V. Betz, "Fast voltage transients on FPGAs: Impact and mitigation strategies," in *Proc. IEEE 27th Annu. Int. Symp. Field-Programmable Custom Comput. Mach. (FCCM)*, San Diego, CA, USA, Apr. 2019, pp. 271–279.
[82] S. S. Mirzargar, G. Renault, A. Guerrieri, and M. Stojilovic, "Nonintrusive and adaptive monitoring for locating voltage attacks in virtualized FPGAs," in *Proc. Int. Conf. Field-Programmable Technol. (ICFPT)*, Maui, HI, USA, Dec. 2020, pp. 288–289.
[83] 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 *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, Munich, Germany, Nov. 2021, pp. 1–9.

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

BEATRICE SHOKRY received the B.Sc. degree in electronics and communications engineering with a minor in computer science from The American University in Cairo, Egypt. She is currently pursuing the Ph.D. degree with EPFL, Lausanne. She has been an undergraduate Research Assistant. She was also a Summer@EPFL Intern, in 2022. Her research interests include hardware security, fault tolerance, and FPGAs.

VINCENT LENDERS (Member, IEEE) received the M.Sc. and Ph.D. degrees in electrical engineering and information technology from ETH Zürich, in 2001 and 2006, respectively. After the Ph.D. degree, he was a Postdoctoral Researcher with Princeton University. In 2008, he joined armasuisse, where he is currently the Director of the Cyber-Defense Campus. His research interests include 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 he has contributed to the development of various cyber security and information systems which have been adopted by the Swiss Federal Department of Defense. He is also the Co-Founder and a member of the Board of the OpenSky Network and Electrosense Associations.

WEI HU (Member, IEEE) received the B.S., M.S., and Ph.D. degrees from Northwestern Polytechnical University (NPU), Xi'an, China, in 2005, 2008, and 2012, respectively. He is currently a Professor with the School of Cybersecurity, NPU. He has published over 70 papers in peer-reviewed journals and conferences, two books, and six patents. His research interests include hardware security, cryptography, formal security verification, logic and high-level synthesis, formal methods, and reconfigurable computing. He serves as a Guest Associate Editor for IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS. He has been an Organizing Committee Member of IEEE International Symposium on Hardware Oriented Security and Trust and Asian Hardware Oriented Security and Trust Symposium, since 2017. He was the Technical Program Co-Chair of 2019 Asian Hardware Oriented Security and Trust Symposium (AsianHOST), the General Co-Chair of 2023 AsianHOST, and a Technical Program Committee Member of ICCD, ASAP, and CFTC.

MIRJANA STOJILOVIĆ (Senior Member, IEEE) received the Dipl.-Ing. and Ph.D. degrees from the School of Electrical Engineering, University of Belgrade, Belgrade, Serbia, in 2006 and 2013, respectively. Since 2016, she has been with the School of Computer and Communication Sciences, EPFL, Lausanne, Switzerland. Her research interests include electronic design automation, reconfigurable computing, and hardware security. She is a Principal Investigator with the Swiss National Science Foundation (SNSF)-funded project Secure FPGAs in the Cloud. She serves on the technical program committees of the International Symposium on Field-Programmable Gate Arrays (FPGA), International Symposium on Field-Programmable Custom Computing Machines (FCCM), International Conference on Field-Programmable Logic and Applications (FPL), and Design, Automation and Test in Europe (DATE) Conference. She is an Associate Editor of the *ACM Transactions on Reconfigurable Technology and Systems* (TRETS) and IEEE EMBEDDED SYSTEMS LETTERS (ESL).