# Remote and Stealthy Fault Attacks on Virtualized FPGAs

Jonas Krautter, Dennis R. E. Gnad, Mehdi B. Tahoori
*Chair of Dependable Nano Computing (CDNC)*
*Karlsruhe Institute of Technology (KIT)*
Karlsruhe, Germany
{jonas.krautter, dennis.gnad, mehdi.tahoori}@kit.edu

***Abstract*—The increasing amount of resources per FPGA chip makes virtualization and multi-tenancy a promising direction to improve utilization and efficiency of these flexible accelerators in the cloud. However, the freedom given to untrusted parties on a multi-tenant FPGA can result in severe security issues. Side-channel, fault, and Denial-of-Service attacks are possible through malicious use of FPGA logic resources. In this work, we perform a detailed analysis of fault attacks between logically isolated designs on a single FPGA. Attacks were often based on mapping a massive amount of Ring Oscillators into FPGA logic, which naturally induce a high current and subsequent voltage drop. However, they are easy to detect as combinational loops and can be prevented by a hypervisor. Here, we demonstrate how even elaborate fault attacks to recover a secret key of an AES encryption module can be deployed using seemingly benign benchmark circuits or even AES modules themselves to generate critical voltage fluctuations.**

***Index Terms*—FPGA, DFA, Fault, Attack, On-Chip, Remote, Multi-user, Multi-tenant, Cloud, AES**

## I. Introduction

Field Programmable Gate Arrays (FPGAs) have become an important component for accelerating emerging applications such as AI workloads from the cloud to the edge. Major cloud computing providers [1]–[3], have added FPGAs to their portfolio and reconfigurable logic is embedded with hard processors into Systems on Chip (SoCs). Therefore, the virtualization of reconfigurable logic resources and multi-tenant access to FPGAs is of increasing interest, allowing higher utilization and efficiency [4].

Fault attacks have been introduced more than 20 years ago [5] and are able to break cryptographic implementations without the need for finding any algorithmic weakness. Traditionally, they require physical access by an attacker to the target hardware. However, this restriction has already been softened by novel attacks such as the Rowhammer attack, where bitflips in memory can be induced remotely [6], [7].

Due to the characteristics of on-chip Power Distribution Networks (PDNs), switching activity on the chip leads to supply voltage fluctuations [8], which is not only a reliability but also a security issue when it comes to FPGA multi-tenancy. Activity of a small fraction of FPGA logic can cause a sufficiently excessive voltage drop to crash an entire FPGA, together with an integrated CPU, if the correct pattern of switching activity is applied [9]. More recently, it was demonstrated how the generation of voltage fluctuations can be carefully calibrated to inject faults exactly into the penultimate round of an AES encryption to fully recover the secret key within a few seconds or minutes [10]. In this initial work, Ring Oscillators (ROs) were used to maliciously manipulate power consumption in a way to impact supply voltage. Follow-up works have shown that it is possible to cause timing faults not only with ROs, but also with sequential oscillators [11], memory access collisions [12], or even by toggling a large amount of AES modules [13]. In other related work, it has been shown that voltage fluctuations inside an FPGA can be measured using the existing FPGA primitives, which allows to perform remote power analysis attacks [14], [15].

In this work, we provide an extensive analysis of how attackers are able to induce timing faults and perform elaborate fault attacks by using seemingly benign logic for generating voltage fluctuations. We deploy key recovery attacks on an AES module and evaluate the attack on a broader selection of devices from different manufacturers. Moreover, we discuss how devices are affected differently and draw conclusions for improving attacks as well as countermeasures.

The remainder of the paper is structured as follows: Section II explains the proposed threat model and the necessary background on how voltage fluctuations occur inside chips. Moreover, we briefly outline the Differential Fault Analysis (DFA) method that is applied in our attacks. Section III details our experiments as well as the used devices and different methods for causing voltage fluctuations. In Section IV, we present results on attacking an AES encryption module, showing key recovery and fault injection rates. We discuss our new findings in Section V and conclude our work in Section VI.

## II. Preliminaries

In this section, we detail the full threat model that we consider for remote on-chip fault attacks. Moreover, we briefly explain the theoretical background of our experiments and the basics of causing a voltage drop and subsequent timing faults.

### A. Threat Model

First, we further describe the attacker-victim scenario assumed throughout this paper. A brief overview of this scenario is given in Fig. 1. We assume the victim and the adversary to have access to a fraction of an FPGA, in which they can load their own arbitrary design, like a cryptographic accelerator.

978-3-9819263-5-4/DATE21/©2021 EDAA

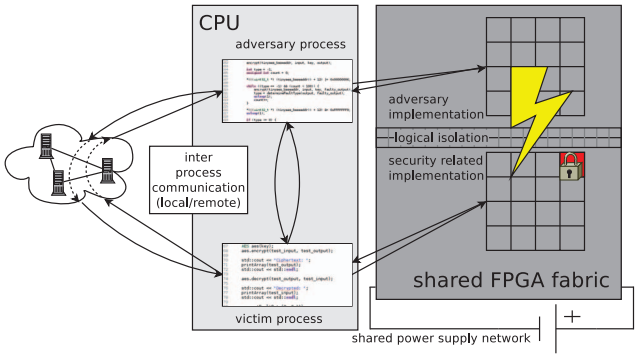



Fig. 1: Overview of the threat model as shown in [10]: Attacker and victim share an FPGA resource with a common power supply network, but isolated, logically disconnected partitions on the fabric

Both attacker and victim have their respective software processes in an operating system, and their designs on the FPGA are logically and spatially separated, and follow other common best practices as explained in [16]. It is also assumed that the respective FPGA fabric is powered by a single common power supply. This scenario includes both data-center applications, in which FPGAs are utilized as standalone accelerators, as well as SoC platforms, in which multiple processes on the CPU can utilize a fraction of the FPGA logic.

We assume the victim to utilize their part of the programmable logic for a security related algorithm, such as a block cipher. A secret key used in this algorithm is either hard-coded onto the FPGA or transferred at runtime. The attacker is able to interact with the victim through a public interface, for instance, to issue encryption requests.

### B. DFA on AES

As a proof-of-concept for successfully conducting an on-chip fault attack on a cryptographic implementation, we evaluate a DFA method on an FPGA implementation of the block cipher Advanced Encryption Standard (AES). In this work, we attack an implementation with 128 bit key length.

DFA is based on causing the same plaintext to be encrypted twice – the first time to gain the correct ciphertext and the second time to acquire a faulty ciphertext as the result of fault injection at a specific point of the algorithm. The ciphertext pairs, each consisting of a correct and a faulty ciphertext of the same plaintext, are then evaluated to extract information about the secret key of the cipher.

In 2003, a generic fault attack on Substitution-Permutation Network (SPN) ciphers was introduced, which only requires two ciphertext pairs to recover the original secret key [17]. We apply this attack with multiple adaptions for practical feasibility to use it on an AES module implemented on an FPGA, and therefore elaborate on this method in detail.

The fault model of the attack is a single random byte fault on a single byte occurring before the 8th round of the AES algorithm. This single byte fault causes all of the output bytes to be incorrect and allows to recover the secret encryption key

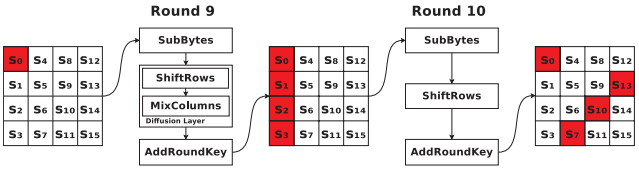



Fig. 2: Propagation of a faulty byte in the input of the 9th round of the AES algorithm as shown in [10]

with only two ciphertext pairs. However, we make use of the distinct fault propagation in the AES encryption to calibrate our fault injection parameters, which is explained later. As depicted in the example in Fig. 2, a single byte fault on the first byte of the state matrix before the penultimate encryption round (round 9) is propagated and results in four faulty bytes at specific positions in the output ciphertext: 0, 7, 10 and 13. Checking for the specific positions of incorrect bytes in the output ciphertext allows us to verify, whether a fault was actually injected before the 9th round, or occurred earlier or later in the computation.

In [17], the number of candidates remaining after two ciphertext pairs was only one (the correct) candidate in 98% of all cases. In the other 2% of cases, only two or a maximum of four key candidates were left. Therefore, to recover the entire round key of the last round with faults injected before round 9, we need a minimum of eight ciphertext pairs: For each of the four key bytes, two pairs are needed. As the key scheduling algorithm is reversible, we can simply compute the original secret key after recovering the last round key.

### C. Fault Injection using FPGA logic

The clock signal used for any functional block in a synchronous FPGA design leads to respective timing constraints for all paths of that design. Typically, the FPGA mapping tool fulfills these timing constraints under certain assumptions of process variation, and runtime variations of voltage and temperature. However, extreme voltage glitches can thus still cause timing faults and wrong results being captured in sequential registers.

To understand how the supply voltage of an FPGA can be decreased with on-chip logic elements, it is necessary to understand how the PDN of an FPGA behaves under the influence of different designs on the fabric. The PDN includes a network from the voltage regulation module on the board down to the internal power rails and every transistor on the FPGA. Generally, the PDN can be modelled as a mesh of resistive, inductive and capacitive elements. Therefore, the power supply voltage depends on two parameters: The average static current drawn by the implemented design (IR-drop) and the voltage drop caused by switching activity and inductance ($di/dt$-drop). The relation is summarized in the *Law of Inductance*: $V_{\text{drop}} = IR + Ldi/dt$. With technology scaling, the effect of static voltage drops has become less relevant compared to the voltage fluctuations caused by switching activity and inductive components [18].

To evoke malicious switching activity on an FPGA and cause an excessive $di/dt$-drop, a certain amount of logic must be enabled in a specific pattern. In [9], ROs were used to induce

voltage drops high enough to crash FPGA-based systems. A singular activation of a large amount of ROs was shown to cause the voltage to drop rapidly by a certain amount and then more slowly return to the original value within about $50$ $\mu$s for the tested devices. Moreover, it was described, how a drop can be increased significantly, by driving the entire grid of oscillators with a different, slower frequency, constantly enabling and disabling the ROs. A dependency on the duty-cycle of this RO toggle signal was demonstrated as well [19].

In this work, instead of crashing the device a lower amount of logic is used and the exact parameters – toggle frequency and duty-cycle – are determined such that the voltage exceeds operating limits at a specific point in time, namely a specific round in the AES encryption. Note that the attacker design must be designed in a way that faults do not accidentally stop the attack somehow, i.e. with a high timing margin. In addition to the attack presented in [10], we also show similar to previous works [13], that the RO grid can be almost arbitrarily replaced by seemingly benign designs such as AES modules or benchmark circuits.

## III. Experiments

Here, we explain the detailed attack setups on different devices and how parameters are automatically calibrated for precise fault injection during a specific AES encryption round.

### A. General Setup and Devices

The attacks are performed on devices from two different manufacturers, namely FPGAs and FPGA-SoCs from Intel/Altera and Lattice Semiconductor. In all cases, two logically and spatially separated designs are deployed on the FPGA fabric, whereas the software part of the threat model is realized as a single process for simplicity. Where a hard processor is available in the FPGA-SoCs, we run the software part inside the Linux system, running on that integrated processor. The smaller devices from Lattice Semiconductor are attached through a serial interface, where the software part is running on the host computer.

The AES implementation used as a proof-of-concept in this work is a simple, small module for $128$ bit key length encryption. It utilizes around $300 - 400$ registers and about $750 - 850$ Look-Up Tables (LUTs) in the tested FPGAs and takes $50$ clock cycles to encrypt a given plaintext. The module is not protected against side-channel or fault attacks.

Development tools for synthesis as well as placement and routing from all vendors offer tools for Static Timing Analysis (STA), which analyzes the design in terms of timing violations under different models (corners) for a given device with a specific speed grade. We evaluate both designs where no timing violations are reported by the analysis, which we refer to as constrained designs, as well as unconstrained designs, where timing violations are reported, but the encryption module still works as expected. Naturally, attacks on fully constrained designs are more difficult, as the margin for a timing fault to occur is much higher. In our results, we show how on some devices only unconstrained designs can be attacked, whereas on other devices even modules, which do not violate timing constraints according to the vendor tools, can be attacked. Table I provides an overview on the platforms where we implemented our attack.

| Vendor | Board | Device |
|---|---|---|
| Lattice Semiconductor | iCE40HX8K-B-EVN | iCE40HX8K |
| Intel | Terasic DE1-SoC | Cyclone V SoC |
| Intel | Terasic DE0-Nano-SoC | Cyclone V SoC |
| Intel | Terasic DE10-Pro | Stratix 10 SX SoC |

TABLE I: Overview of the evaluated platforms

### B. Automated Parameter Calibration

As described in Section II-B, parameters such as frequency and duty-cycle for toggling the malicious logic need to be adapted to provoke faults at the proper moment of the encryption. To make use of the possibility for injection success verification when injecting before the 9th encryption round, we develop an automated calibration algorithm, to be executed before evaluating injection rates or attack success for a given design and device. The algorithm allows to use the attacker design in different setups, without the need for finding appropriate parameters in time-consuming trial-and-error experiments manually.

We adapt the signal for activating the power wasters in three parameters: The toggle frequency, the duty-cycle and the delay between starting the encryption and activating the toggling. The algorithm performs as follows:

a) The attacker activates the calibration process on the FPGA. A random input plaintext is encrypted without malicious activity. The result is stored as the correct ciphertext.
b) Afterwards the fault injection on the FPGA is activated, to enable switching activity for the following encryptions.
c) Encryption of the same random plaintext is requested. The attacker design on the FPGA activates the power wasters with an initial frequency and duty-cycle and no activation delay. If no fault is detected, the frequency/duty-cycle are decreased/increased. If a fault is detected, which affects an undesired subset of bytes, the injection occurred too early or too late, and the activation delay is increased/decreased. In any case, the attacker design reports the injection success to the attacker process, which either requests another encryption or continues the process.
d) If the injection was successful or a predefined maximum of injection attempts $inj_{max}$ were unsuccessful, the attacker software deactivates its corresponding FPGA design and either finishes the successful scan or chooses another random plaintext for fault injection.

### C. Power Wasting Circuits

Whereas the initial work on chip-internal fault attacks [10] made use of a large RO-grid to generate voltage fluctuations, we explore two additional options in this work. The main objective is to investigate whether benign-looking logic can be used to inject precise faults in the threat model of multi-tenant FPGAs. In [13], among other approaches multiple AES encryption rounds performing random computations at a very high clock frequency are used to inject timing faults into a ripple-carry

adder design. We confirm that AES modules can be employed as power wasters in our attack setup as well and also successfully evaluate the *s1238* circuit from the ISCAS'89 benchmark collection [20] to inject faults with seemingly benign logic. The *s1238* is described as a combinational circuit with random flip-flops in [20], which we assume to be beneficial in terms of switching activity, unlike, for instance, a comparatively static state machine circuit.

For the RO-based attack, the oscillators are implemented with a single LUT using various Verilog keywords to keep the synthesis software from optimizing the grid away. On Intel/Altera FPGAs this can be achieved by declaring virtual output pins, which are placed as fanless LUTs on the chip. For the Lattice Semiconductor toolchain, using the *keep* or *syn_keep* directives is enough to prevent optimization. The ROs can be enabled or disable through a global signal, which – on Intel/Altera FPGAs – is routed on global clock routing resources to prevent long compile times and congestion.

To inject faults with AES modules or the *s1238* benchmark circuit, we configure a Phase Locked Loop (PLL) on the chip to generate a 750 MHz clock signal, driving the respective registers without regard for any timing violations that might occur. As with the RO grid, a global toggle signal is used to switch the instances on or of with the correct parameters that have been determined by the calibration algorithm explained in the previous Section III-B. Again, any optimization is prevented through the use of Verilog keywords and virtual pin instantiations. A major difference to injecting faults with ROs is the choice of input values for the power wasters, which is not trivial as the switching of gates and registers inside the designs should be maximized. For the AES modules, computing an encryption stream where the output ciphertexts are fed back as the new input to the circuits is sufficient. This is most likely due to their well distributed switching behaviour, as a good encryption algorithm should be generating ciphertexts that cannot be distinguished from random output. The *s1238* from the ISCAS'89 benchmark is initially driven with a zero input, then we calculate the new input value $\text{in}_n$ at clock cycle $n$ as $\text{in}_n = \overline{\text{in}_{n-1}} \oplus (\text{out}_{n-1} << 1))$. Intuitively, this approach initially flips the maximum amount of bits in the input value, but also takes the circuit output into account to prevent the circuit from getting stuck in a specific state.

## IV. Results

Our main experimental platform is the Terasic DE1-SoC board, which incorporates an Intel/Altera Cyclone V FPGA-SoC. On this board we present fault injection rates and key recovery statistics as shown in [10] as well as the additional results using AES modules and benchmark circuits for causing voltage fluctuations. Lastly, we present results on the iCE40-HX8K FPGA from Lattice Semiconductor as well as the Intel/Altera Stratix 10 SoC on the Terasic DE10-Pro board.

### A. Fault Injection Rate

In order to evaluate the general fault injection efficiency, we first generate bitstreams for different percentages of logic utilization of the attacker logic in the range of 30% to 50% for the DE1-SoC board. The victim module runs at a frequency of 111 MHz, which does not violate any timing constraints as explained in the previous subsection, even in the worst-case corner of the STA (slow 1.1V/85°C model). Then we measure the number of faults occurring for one million encryption requests from a previously generated set of random plaintexts, which are reused for all experiments. The encryption key remains the same for one test series, which is repeated for a second random key.

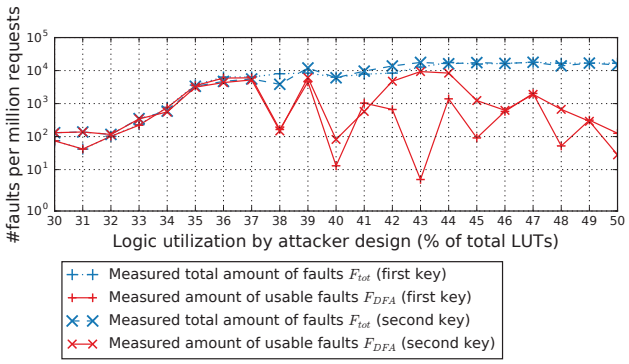



Fig. 3: Total measured fault injection rates $F_{\text{tot}}$ and measured injection rate of faults usable in DFA $F_{\text{DFA}}$ with respect to the amount of logic utilization (percentage of total LUTs) by the attacker design for two different random encryption keys

Fig. 3 shows the total number of faults out of 1M trials $F_{\text{tot}}$ as well as the number of faults usable for DFA with our described fault model $F_{\text{DFA}}$ depending on how much of the available logic resources is occupied by the attacker design. We see that both $F_{\text{tot}}$ and $F_{\text{DFA}}$ initially increase at the same rate. This proves the effectiveness of the calibration algorithm before each evaluation. However, if the amount of activated ROs exceeds a certain level, the effect is too strong to allow precise injections. $F_{\text{tot}}$ still increases with more ROs, but can affect more than one round or byte per round. Hence, the resulting ciphertext will have more than four byte faults, which cannot be used anymore to recover the secret AES key in the used fault model.

We repeated the experiment on three Terasic DE1-SoC boards of different age and usage history with similar results. The minimum and maximum amount of logic that allows ideal injection rates is slightly different for each board, but a common suitable region that covers all devices can be determined.

### B. Key Recovery Success Rate

Subsequently, we evaluate the success of the full DFA attack including recovery of the secret AES key. This evaluation reflects on the success of our entire algorithmic flow of injecting faults before the 9th AES encryption round, the calibration algorithm and filtering of undesired faulty ciphertexts. For each of the three boards, which we already used to investigate fault injection rates, we use the amount of ROs that lead to the highest injection rate $F_{\text{DFA}}$ of faults usable for DFA. We generate a set of 5 000 random AES keys and collect a minimum of two ciphertext pairs, which exhibit faults at the desired positions, for each four bytes of the last AES round

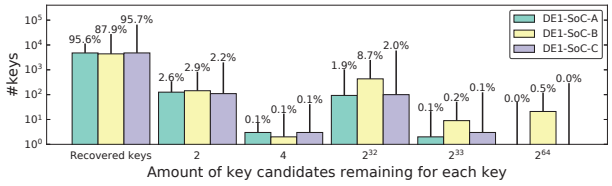



Fig. 4: Amount of key candidates remaining for each key recovery attempt on 5 000 random AES keys on three different DE1-SoC boards, using ROs to cause voltage fluctuations.

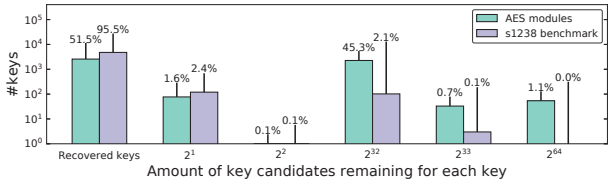


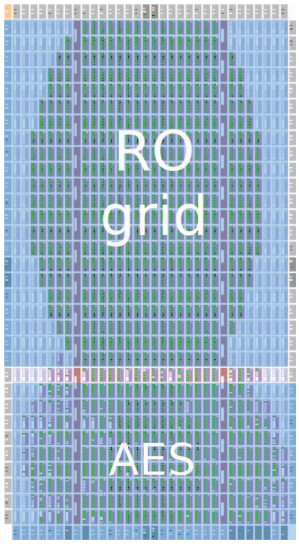

Fig. 5: Amount of key candidates remaining for each key recovery attempt on 5 000 random AES keys, using AES modules or *s1238* benchmark circuits to cause voltage fluctuations.

key. The ciphertexts along with each key are stored on the SD card of the board and later transferred to a host computer.

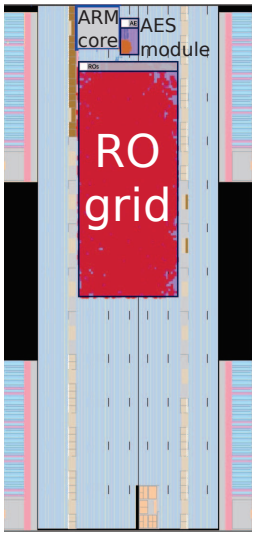
Fig. 4 summarizes the results of the key recovery attempts on our three DE1-SoC boards. On all three boards, we are able to deploy the attack successfully recovering all 16 key bytes correctly for at least 87.9% of the 5 000 random keys. All recovered keys are correctly recovered, so no false positives are encountered. On all boards, we have a small amount of around 2−3% of all keys which cannot be recovered, but less than four candidates for the last round key remain. This ratio confirms the results in [17], showing that in about 2% of the cases more than two ciphertext pairs are necessary to recover the AES key. If a sufficiently small amount of key candidates remains, the correct key can be easily recovered with an exhaustive search. We encounter, however, some keys, where more than $2^{32}$ or even $2^{64}$ candidates remain. Across all our experiments, an average of 22 usable faults were required to gather the required two ciphertext pairs per four bytes of the round key. To collect these pairs, the attacker design needs to issue 17 979 encryption requests on average to the AES module, which took on average 2 344 ms. The average time for the evaluation of one attack until key recovery on the described host machine is about 107 ms.

### C. Using Benign Logic for Fault Injection

To extend on the original results from [10], we perform fault attacks using seemingly benign circuits described in Section III-C. We perform the attacks on the Terasic DE1-SoC, replacing the previously used ROs with either AES modules or the ISCAS'89 *s1238* benchmark circuit. For successful key recoveries, we need 60 AES modules or 280 benchmark circuits, corresponding to 50.1% and 64.7% of the available logic resources each. The larger amount of logic required when using the *s1238* is most likely due to the higher amount of switching occurring when operating AES modules. In both cases, the instances are operated at a frequency of 750 MHz.




(a) Floorplan of the design on the Lattice iCE40-HX8K FPGA




(b) Floorplan of the design on the Stratix 10 SX SoC

Fig. 6: Evaluation designs on devices from different manufacturers as seen in the respective floorplanning tools.

In Fig. 5, we again present results of attempting to recover 5 000 randomly drawn secret AES keys using seemingly benign logic for fault injection. Interestingly, fewer keys can be recovered when attempting the attack using AES modules as power wasters for generating voltage fluctuations, whereas the recovery rate when using the *s1238* circuits is very close to an RO-based attack. We observe a high amount of keys, where $2^{32}$ candidates remain, when using AES modules. Since faults that are injected too early or too late are filtered, the only valid conclusion is that more multi-byte faults are being injected in this case. A possible explanation is the size of the AES modules, compared to the *s1238* modules or ROs, which may result in voltage spikes of slightly longer duration thus affecting multiple bytes.

### D. Experiments on Different Platforms

We also confirm that the attack is successful on different platforms from different manufacturers. In addition to Cyclone V devices, we present results on an FPGA from Lattice Semiconductor, as well as a server-grade Stratix 10 SX SoC on the Terasic DE10-Pro board.

In Fig. 6, we present the evaluation designs on the two additional devices. On the Stratix 10, about 12% of the available logic resources are used for the RO grid (115 000 ROs), whereas on the small low-power FPGA from Lattice Semiconductor the attacker uses exactly 50% of the 7 680 available LUTs. Empirically we find this to be the maximum amount of ROs that can be deployed with a single enable signal on the Stratix 10 device, otherwise the board crashes during programming. Whereas a more elaborate toggle signal design may enable to deploy a larger amount of oscillators, the amount is already sufficient to inject faults into an AES module running at 255 MHz. In this setup, the AES module is reported as constrained

in the fastest timing corner and works without errors when the attacker design is disabled.

On the Lattice iCE40-HX8K FPGA, the AES module operates fully constrained at 24 MHz and we are able to recover the secret AES key within a few seconds. Increasing the amount of ROs on this board, we are also able to cause a complete crash, where only a power-cycle can re-enable the device. An important observation here is that the crash seems to affect the PLL of the devices, whereas purely combinational logic or externally clocked registers remain functional. Further investigation on which exact components are affected by voltage fluctuations may be an important topic for future work and could result in effective countermeasures.

## V. Discussion

The results presented here make the development of countermeasures more critical but also more difficult. Recently proposed solutions [21], [22] based on checking bitstreams for potentially malicious signatures may need to be adapted to be able to detect injection logic that makes use of AES modules or other benign circuits for injection. More generic approaches, such as the estimation of a design's maximum switching activity could overcome the need for continuously introducing new virus signatures, but will be computationally expensive.

However, experiments on different platforms show, that the switching activity may affect different parts of the devices. When crashing the Lattice iCE40-HX8K FPGA the PLL is affected, whereas purely combinational logic or flip-flops driven by external oscillators can glitch but will still be responsive. Potential future countermeasures may be based around creating exclusive voltage lanes for critical components of the FPGA, such as the clock generators or hypervisor logic.

On the attacker side, we prove how the selection of input values to benign logic that is used for creating voltage fluctuations is critical to the attack success. Whereas we empirically select inputs based on successful fault injection, a more elaborate method would be to find an input sequence leading to the maximum amount of switching gates in a specific module. Although finding such an input sequence may need exponential computational effort depending on the size of the circuit, it can discover critical inputs even in non-reconfigurable hardware.

design. The results emphasize the importance of developing countermeasures to enable secure FPGA virtualization.

## VI. Conclusion

In this work, we demonstrate how elaborate remote fault attacks in multi-tenant FPGAs do not require an attacker to use highly specialized logic and can be deployed on devices from different manufacturers. With seemingly benign logic circuits the attacker is still able to cause voltage fluctuations to inject timing faults into an AES hardware encryption module on FPGAs and FPGA-SoCs from Intel/Altera and Lattice Semiconductor, proving both embedded as well as cloud devices vulnerable. Using a calibration mechanism to inject faults exactly before the 9th AES encryption round, it is possible to recover up to 90% of secret AES keys on the Cyclone V FPGA-SoC, without any logical connection between attacker and victim

## References


[1] Amazon EC2 F1 instances. [Online]. Available: https://aws.amazon.com/ec2/instance-types/f1/

[2] Intel FPGAs power acceleration-as-a-service for alibaba cloud. [Online]. Available: https://newsroom.intel.com/news/intel-fpgas-power-acceleration-as-a-service-alibaba-cloud

[3] Huawei Cloud. FPGA accelerated cloud server. [Online]. Available: https://www.huaweicloud.com/en-us/product/fcs.html

[4] A. Vaishnav, K. D. Pham, and D. Koch, "A survey on FPGA virtualization," in *28th International Conference on Field Programmable Logic and Applications (FPL)*, 2018, pp. 131–1317.

[5] D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the importance of checking cryptographic protocols for faults," in *Advances in Cryptology — EUROCRYPT '97*, 1997, pp. 37–51.

[6] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors," in *International Symposium on Computer Architecture (ISCA)*, 2014, pp. 361–372.

[7] D. Gruss, C. Maurice, and S. Mangard, "Rowhammer.js: A remote software-induced fault attack in javascript," *CoRR*, 2015.

[8] K. M. Zick, M. Srivastav, W. Zhang, and M. French, "Sensing nanosecond-scale voltage attacks and natural transients in FPGAs," in *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays - FPGA '13*, 2013.

[9] D. R. E. Gnad, F. Oboril, and M. B. Tahoori, "Voltage drop-based fault attacks on FPGAs using valid bitstreams," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, 2017.

[10] J. Krautter, D. R. E. Gnad, and M. B. Tahoori, "FPGAhammer: remote voltage fault attacks on shared FPGAs, suitable for DFA on AES," *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*, vol. 2018, no. 3, 2018.

[11] T. Sugawara, K. Sakiyama, S. Nashimoto, D. Suzuki, and T. Nagatsuka, "Oscillator without a combinatorial loop and its threat to FPGA in data centre," *Electronics Letters*, vol. 55, no. 11, pp. 640–642, 2019.

[12] M. M. Alam, S. Tajik, F. Ganji, M. Tehranipoor, and D. Forte, "Ram-jam: Remote temperature and voltage fault attack on fpgas using memory collisions," in *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, 2019, pp. 48–55.

[13] G. Provelengios, D. Holcomb, and R. Tessier, "Power wasting circuits for cloud FPGA attacks," in *30th International Conference on Field-Programmable Logic and Applications (FPL)*, 2020, pp. 231–235.

[14] F. Schellenberg, D. R. Gnad, A. Moradi, and M. B. Tahoori, "An inside job: Remote power analysis attacks on FPGAs," in *Proceedings of Design, Automation & Test in Europe (DATE)*, 2018.

[15] M. Zhao and G. E. Suh, "FPGA-based remote power side-channel attacks," in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 805–820.

[16] T. Huffmire, B. Brotherton, G. Wang, T. Sherwood, R. Kastner, T. Levin, T. Nguyen, and C. Irvine, "Moats and drawbridges: An isolation primitive for reconfigurable hardware based systems," in *2007 IEEE Symposium on Security and Privacy (SP '07)*, 2007.

[17] G. Piret and J.-J. Quisquater, "A differential fault attack technique against SPN structures, with application to the AES and khazad," in *Cryptographic Hardware and Embedded Systems (CHES)*, 2003.

[18] K. Arabi, R. Saleh, and X. Meng, "Power supply noise in SoCs: Metrics, management, and measurement," *IEEE Design & Test of Computers*, vol. 24, no. 3, pp. 236–244, 2007.

[19] D. Gnad, F. Oboril, S. Kiamehr, and M. Tahoori, "An experimental evaluation and analysis of transient voltage fluctuations in FPGAs," *IEEE Annals of the History of Computing*, no. 10, pp. 1817–1830, 2018.

[20] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *IEEE International Symposium on Circuits and Systems, 1989*, May 1989, pp. 1929–1934 vol.3.

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

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