X-Attack: Remote Activation of Satisfiability Don't-Care Hardware Trojans on Shared FPGAs

Dina G. Mahmoud*, Wei Hu‡ and Mirjana Stojilović*
* EPFL, Lausanne, Switzerland
‡ Northwestern Polytechnical University, Xi'an, China
Email: dina.mahmoud@epfl.ch, weihu@nwpu.edu.cn, mirjana.stojilovic@epfl.ch

***Abstract*—Albeit very appealing, FPGA multitenancy in the cloud computing environment is currently on hold due to a number of recently discovered vulnerabilities to side-channel attacks and covert communication. In this work, we successfully demonstrate a new attack scenario on shared FPGAs: we show that an FPGA tenant can activate a dormant hardware Trojan without any physical or logical connection to the private Trojan-infected FPGA circuit. Our victim contains a so-called *satisfiability don't-care* Trojan, activated by a pair of don't-care signals, which never reach the combined trigger condition under normal operation. However, once a malicious FPGA user starts to induce considerable fluctuations in the on-chip signal delays—and, consequently, the timing faults—these harmless don't-care signals take unexpected values which trigger the Trojan. Our attack model eliminates the assumption on physical access to or manipulation of the victim design. Contrary to existing fault and side-channel attacks that target unprotected cryptographic circuits, our new attack is shown effective even against provably well-protected cryptographic circuits. Besides demonstrating the attack by successfully leaking the entire cryptographic key from one unprotected and one masked AES S-box implementation, we present an efficient and lightweight countermeasure.**

***Index Terms*—SDC hardware Trojans, FPGA, multitenancy, timing faults, remote attack**

I. Introduction

Today, heterogeneous cloud computing platforms allow cloud users to gain almost full access over the low-level logical and electrical features of large FPGA instances remotely [1], [2]. At the same time, partial reconfiguration enables temporal and spatial multiplexing of the FPGA die, rendering a multi-tenant use mode compatible with the rest of the cloud ecosystem. These appealing computing features, however, have raised new security concerns supported by recent denial-of-service, fault-injection, power side-channel and crosstalk side-channel attacks, as well as covert communication on shared FPGAs [3]–[10].

In this paper, we present a new attack vector on shared FPGAs, which combines remotely-induced timing faults and hardware Trojans. The Trojans studied here are triggered using a pair of carefully chosen don't-care signals, commonly represented as `X`; hence the name of `X`-attack. They can be inserted into a multitude of third-party IP cores. Such hardware Trojans can be both hard to detect and eliminate, because they hide behind don't-care states. Moreover, by construction, these Trojans can leak the secret directly through publicly observable core outputs, bypassing the common protections against side-channel attacks. We demonstrate that a malicious FPGA tenant (the attacker) can trigger a don't-care Trojan without any physical or logical connection to the Trojan-infected design (the victim). Using two designs as targets—one pipelined AES core and one masked AES S-box—we show how an attacker can extract the entire encryption key by merely observing the statistical distribution of faulty ciphertexts under a remotely-controlled `X`-attack. Additionally, we design a countermeasure and demonstrate its effectiveness in protecting against this attack.

The remainder of this paper is organized as follows. Section II reviews related work, while some background is covered in Section III. We describe our threat model in Section IV and the system design in Section V. Experimental evaluation is presented in detail in Section VI. We discuss countermeasures and provide our solution in Section VII. Section VIII concludes the paper.

II. Related Work

A. Attacks in Multi-tenant FPGAs

Many researchers have looked at the possibility of carrying out attacks in multi-tenant FPGAs. Gnad et al. were the first to utilize the FPGA programmable fabric to create malicious voltage drops [4] and demonstrate a denial-of-service attack. Multi-tenant power side-channel attacks have been investigated by Schellenberg et al. [11] and Zhao et al. [9]. A power analysis attack on an Amazon EC2 F1 instance [1] was successfully carried out by Glamočanin et al. [7]. Furthermore, Ramesh et al. [10] and Giechaskiel et al. [8] demonstrated a crosstalk-coupling side-channel attack between the neighboring long wires of an FPGA. Mahmoud et al. used ring oscillators (ROs) to bias the output of a colocated true random number generator [6]. Krautter et al. configured the FPGA logic to induce faults in a cotenant cryptographic circuit to be able to carry out a differential fault analysis (DFA) attack and extract the encryption key [5]. In this work, we too configure the FPGA logic to induce faults in a cryptographic circuit colocated on the same FPGA in an isolated region. However, our work assumes the existence of a hardware Trojan in the victim. Additionally, unlike DFA, we do not require fault injection at a specific encryption round nor a replay attack;

Wei Hu received support from the Natural Science Foundation of Shaanxi Province under grant no. 2019JM-244.

our attack leads to the leakage of the secret directly to the circuit output.

B. Don't-Care Hardware Trojans

Several recent research works have leveraged don't-care conditions for hardware Trojans. Fern et al. used unspecified functionality for Trojan insertion [12]. The observation was that some designs are incompletely specified under certain input conditions, which provides designers the flexibility to decide the design behavior under external don't-care conditions. Such Trojans can be hard to detect due to the lack of golden reference in the design specification to check against the Trojan behavior. Nahiyan et al. added floating states into incompletely specified finite state machines and used fault attack to force the design into the Trojan state to perform malicious activities [13]. Krieg et al. demonstrated the possibility of creating a Trojan trigger using don't-care state which is logical `0` in behavioral simulation while logical `1` in hardware implementation [14]. This Trojan can survive from design verification since the design with the Trojan is equivalent to the specification before deployment on hardware. Hu et al. used internal don't-care conditions for malicious design modifications [15]. Their idea was to use internal design states that will never be satisfied (e.g., two signals cannot be logical `1` at the same time) in normal operation as Trojan trigger. Another recent work used the unspecified functionality in an obfuscated design to implement hardware Trojans [16]; it leveraged the fact that the end user usually does not care about the design behavior under incorrect obfuscation keys. Existing don't-care Trojan research usually relies on a strong threat model, assuming physical or logical access to the Trojan design in order to manipulate inputs, inject faults, and probe the Trojan payload. This paper presents a new attack vector, which eliminates such assumptions.

III. Background

A. Satisfiability Don't-Care Hardware Trojan

Satisfiability don't-cares (SDCs) are particular internal design states in digital hardware that are never reached due to correlations of variables. In other words, such internal design states will never appear under any input sequence during normal operation. For a better understanding, let us consider the gate-level implementation of a 2-to-1 multiplexer (MUX) shown in Fig. 1a, whose Boolean function is $O = S \cdot A + \overline{S} \cdot B$.

Two MUX internal signals, n_1 and n_3, are correlated due to the common select line S. As a result, they will never be logical `1` simultaneously. n_1 and n_3 consist of an SDC condition in that the internal design state $(n_1, n_3) = (1, 1)$ will never be observed under normal operation.

The SDC condition shown in Fig. 1a considers signal pairs connected to the same Boolean gate. Such *local* SDCs can be easily identified through satisfiability analysis. A more generalized concept is *global* SDCs, which represent conflicting design states of signal combinations spreading far apart in the hardware design. As an example, when some primary input is logical `0`, some internal signal would never be logical `1` due to path correlation. Unlike *external* don't-care conditions that are caused by incomplete design specification and thus can be fully eliminated, SDCs are widespread in hardware designs even in completely specified functions.

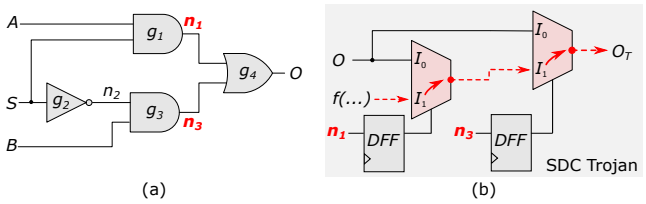


Fig. 1: Example of an SDC condition and an SDC Trojan. (a) An SDC condition in the gate-level implementation of a 2-to-1 multiplexer is shown (signals n_1 and n_3 cannot be logical `1` simultaneously). (b) An SDC Trojan that uses the registered don't-care signal pair to multiplex some function f to the output.

Hu et al. introduced the idea of using an SDC signal pair to create two discrete trigger signals [15]. In Fig. 1b, an example SDC Trojan design is shown; as triggers, it uses the don't-care signals n_1 and n_3 from the MUX implementation. Such SDC Trojan can be hard to detect because the trigger condition can never be satisfied in normal operation: the designs with and without the Trojan are functionally equivalent. In addition, using two discrete trigger signals also protects the Trojan from switching probability analysis because each trigger signal in the Trojan design is able to switch state. Finally, adding flip-flops for the Trojan trigger signals separates the combinational block where the signals originate from that in which the Trojan is implemented. This helps prevent the Trojan being optimized away while not changing design functionality. However, the attack model used by Hu et al. [15] requires physical access to the Trojan implementation in order to overclock the design to trigger the Trojan (by inducing timing failures in the two DFFs and forcing them both to logical `1`) and probe faulty outputs to recover the key. In this work, we consider a more practical attack scenario, which does not assume direct manipulation of signals (e.g., design clock) in an isolated private FPGA region.

B. Remote Timing Fault Attacks

Researchers have shown that FPGA resources can be configured to create internal power supply voltage fluctuations; for example, large banks of fast ROs [4] or glitch-generators [17], if well-controlled, can draw a considerable amount of current and, consequently, cause voltage drops of large amplitude. If excessive, the FPGA core voltage fluctuations created this way can put the chip into reset. However, if the voltage variations are carefully controlled, they can be exploited to cause circuit timing faults: lowering the voltage results in an increased FPGA logic delay, which can lead to a flip-flop setup time violation, metastability, and possibly an incorrect circuit output [18], [19]. These timing-fault attacks were previously leveraged for biasing random number generators [6] and faulting outputs to allow for DFA attacks [5].

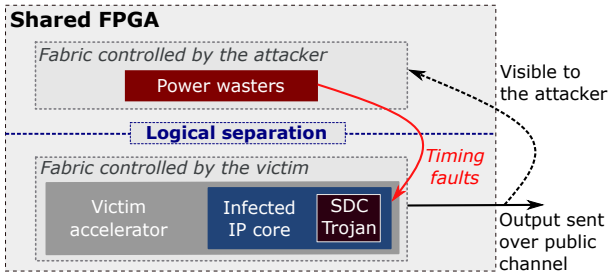


Fig. 2: Threat model overview: the victim and the attacker are colocated on the same FPGA, but physically and logically separated. The victim uses an IP core, which has been infected by an SDC Trojan. Its output is sent over a public channel, allowing the victim to monitor it.

IV. Threat Model

Our threat model is illustrated in Fig. 2. Similarly to other works exploiting hardware Trojans, we assume that the Trojan is inserted into an IP used by the victim and that the attacker has the means to trigger it. We target a shared FPGA scenario, in which the victim and the attacker are colocated on the same FPGA and yet physically and logically separated. They operate independently in private partitions and can load arbitrary designs in their own regions. The victim and the attacker are powered by the same voltage source and share the power-distribution network—a scenario common to both datacenter- and SoC-FPGAs.

In our threat scenario, the victim is running a security-sensitive service (e.g., a symmetric encryption using an AES IP core and a secret key), whose implementation is compromised by an SDC Trojan [15] inserted during the design phase. Even though in our experimental evaluation we assume that the attacker can issue requests to an interface of the victim to initiate encryption and obtain the ciphertexts—such an interface could be accessible locally or remotely via the network—the only true requirement is the ability to observe the victim outputs. The latter would be possible for the outputs sent over a public channel, which is often the case of encrypted data. Finally, we work under the assumption that the victim does not deploy any hardware or software redundancy to verify the correctness of the ciphertexts.

V. System Design

We design and implement the attack on an Intel SoC board [20]. Fig. 3 illustrates the system architecture. The attacker is composed of two banks of ROs and a control circuit. The victim contains a cryptographic module, which receives the plaintext (PT) and sends the ciphertext (CT) to the hard processing system (HPS). For the purpose of observing the voltage variations caused by the attacker, we add to it a delay-line voltage-drop sensor [21]. It is important to note that the sensor is present and used during the initial observation phase only; contrary to power side-channel attacks, we do not rely on the sensor data to extract the secret key, nor do we use it as an indicator of a successful attack after the initial analysis.

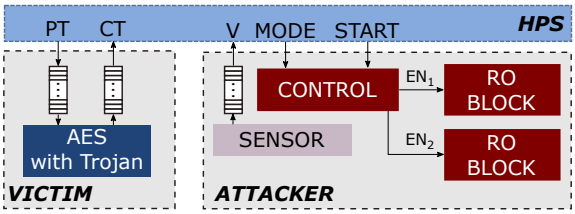


Fig. 3: System block architecture. Hard processing system (HPS) interacts with the victim and the attacker independently.

A. Attacker

Power-wasting circuits are commonly used to increase the victim signal delays and inject timing faults [5], [6]. They can be implemented using ring oscillators, glitch generators, long-wire charging circuits, and even synchronous ROs [17], [22]. Our attacker uses two large equally-sized and independently-controlled banks of ROs and three attack control modes [6]:

1) *periodic*: enable signals are periodic, in phase, having ≈75% duty ratio and the same frequency;
2) *half*: only one enable signal is active;
3) *all*: both enable signals are active.

Besides initiating the attack, the software running on the hard processing system can select the next control mode (MODE in Fig. 3). However, it is the hardware that generates the RO enable signals and limits the duration of the attack. This is to ensure that the attacker does not accidentally send the board to reset, in case of excessively long RO activity.

B. Victim

In general, combining the Trojan with the timing fault attack would target a victim with a secret which can be leaked by the Trojan. This is the case for cryptographic cores commonly used in cloud applications. They have two secrets: the plaintext and the key. Given our threat model, the key would be the desired information to leak; once the key is known, the encryption is compromised.

For this work, we select an openly available 128-bit pipelined AES core [23] and a masked S-box implementation [24]. Using Yosys [25], we look into the S-box of both designs to find a pair of don't-care signals that satisfy the criteria of having only one impossible combination: `11` or `00`. Once that pair is found, we embed the Trojan illustrated in Fig. 1b [15] into the victim.

C. Data Collection

To observe the effects of the attack on the FPGA core voltage, we equip the attacker with a delay-line voltage-drop sensor [4], [6], [18]. The sensor is implemented as a 255-bit ripple-carry adder and uses the carry-chain logic. All bits of the first operand are tied to `0`, while all bits of the second operand are tied to `1`. The carry-in is connected to a clock (running at the same frequency as the attacker system clock, but shifted in phase). When the carry-in is `0`, all the bits of the sum output are normally `1`. Similarly, when the carry-in is `1`, the output bits are normally `0`, and the carry-out is `1`.

However, when the voltage drops significantly, the phase shift between the carry-in and the clock at which the output sum is sampled allows capturing the varying propagation depth of the carry-in through the carry-chain and, consequently, sensing the effect of the attack on the on-chip voltage and delay.

Given the difference in throughput between the software and the hardware, we store all the on-chip data in the FIFO buffers first and then offload them to the HPS for processing. The FIFO operation is controlled from the software.

VI. Experimental Evaluation

We perform the experiments on a Terasic DE1-SoC development board (used in related research as well [5]), with an Intel Cyclone V FPGA and an ARM dual-core Cortex-A9 embedded processor [20]. Using the LogicLock feature in Quartus Standard Edition 19.1, we divide the FPGA fabric in two disjoint regions: one for the attacker and one for the victim, respectively. The ARM processor runs Linux operating system and the control software. In the remainder of this section we describe our experiments on two distinct victims: one unprotected AES core and one masked AES S-box implementation.

A. Unprotected AES FPGA Core Experiments

Using an open-source pipelined AES-128 core [23] as the first victim, we run several groups of experiments, starting with the system validation and calibration. Then, we run the attack and demonstrate that we are able to identify the secret key using the recorded ciphertexts only. Finally, we vary the attacker size and location, to observe their impact on the attack success rate.

1) System Validation and Calibration: To validate the cryptographic core with the SDC Trojan, we feed it with pseudorandom plaintexts and verify that the recorded ciphertexts are always correct, before launching the timing fault attack to activate the Trojan. The AES clock frequency is set to 160 MHz, to satisfy the design critical path delay and the HPS interface constraints.

To verify whether logic synthesis optimizations would result in the Trojan being optimized away, we repeat the synthesis using several optimization options under Quartus and three Trojan variations:

- *Trojan-1*: without modifications (illustrated in Fig. 1b),
- *Trojan-2*: Trojan-1 with inverted polarity triggers, and
- *Trojan-3*: Trojan-2 with extra delay on the trigger path.

For inverted-polarity triggers, we swap the Trojan multiplexer inputs, to guarantee unmodified circuit operation. The results are listed in Table I. Interestingly, none of the optimizations manages to detect and optimize away the Trojans. The reason for three Trojan variations lies in the experimental observation that inverting the don't-care signal polarity has higher chances of triggering the Trojan, possibly thanks to a small amount of added delay. This delay is further increased in Trojan-3, which is convenient during initial attack testing. Moreover, Trojan-3 achieves rather consistent trigger delays across synthesis runs.

TABLE I: Various Quartus optimization efforts. None resulted in the don't-care Trojan being optimized away.

Command	Option	Trojan-1 Remains?	Trojan-2 Remains?	Trojan-3 Remains?
Normal Flow	Balanced	✓	✓	✓
High Effort	Performance	✓	✓	✓
Aggressive	Performance	✓	✓	✓
Aggressive	Area	✓	✓	✓
High Effort	Power	✓	✓	✓

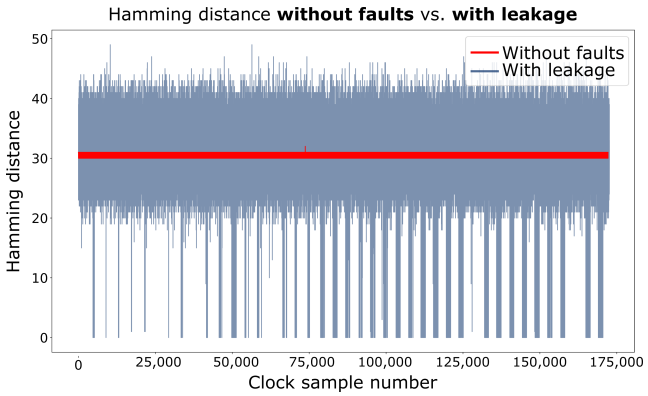
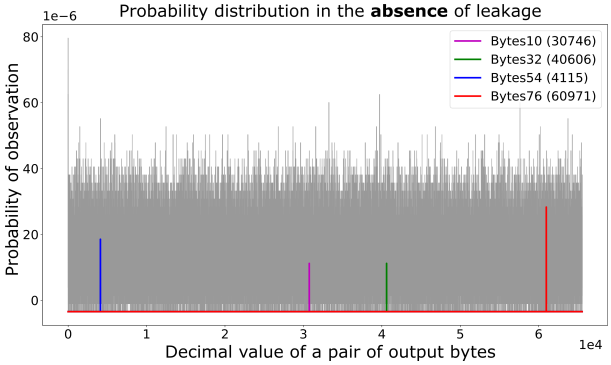


Fig. 4: Hamming distance (HD) between the key and the ciphertext in two experiments. In the first experiment (in red), no faults exist and the HD is approximately 30; in the second (in blue), the key is leaked whenever the HD equals zero.

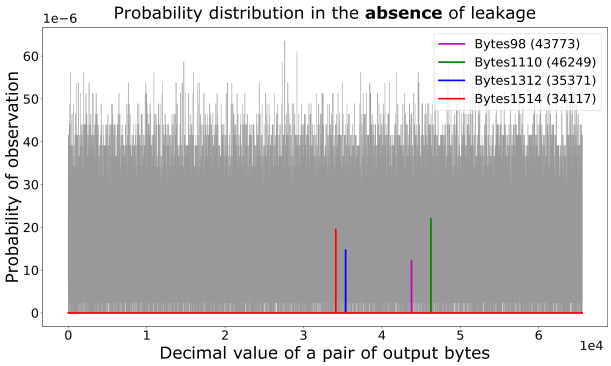
To choose the sequence of RO activation modes, we follow the approach previously demonstrated successful in biasing a true random number generator [6]: first *periodic*, then *half*, and finally *all* mode. However, given the difference in the target FPGA device, we limit the attack duration to a considerably shorter time period and use a smaller number of ROs. Our experiments show that an attacker of 10k–20k ROs—occupying 16%–32% of available adaptive logic modules (ALMs)—can induce the faults in the victim without sending the board to reset. The exact number of ROs to use depends on the target delay increase required to activate the Trojan. This can vary under different relative location configurations of the victim and the attacker. A larger number of ROs can potentially send the board into reset, while a smaller number may fail to increase the signal delays to the point required to cause faults. Finally, we found that an attack duration between 2,048 and 4,096 AES clock cycles (equivalent to 12.8–25.6 µs) is well suited for the target device.

2) Attack Experiments: We combine the victim and the attacker on the FPGA and start testing the attack. In all experiments, we use Trojan-2 and run the attack 100×, each time performing 4,096 encryptions and always pausing for a couple of seconds between two consecutive runs.

When the attack is successful, the key usually leaks more than once. This is illustrated in Fig. 4, by comparing the Hamming distance (HD) between the ciphertext and the known key in two experiments. In both, two fixed plaintexts alternate at the input; the plaintexts are chosen so that, in the absence

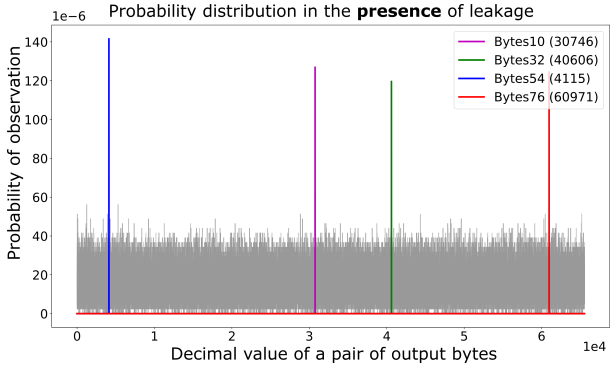


(a) Least significant bytes

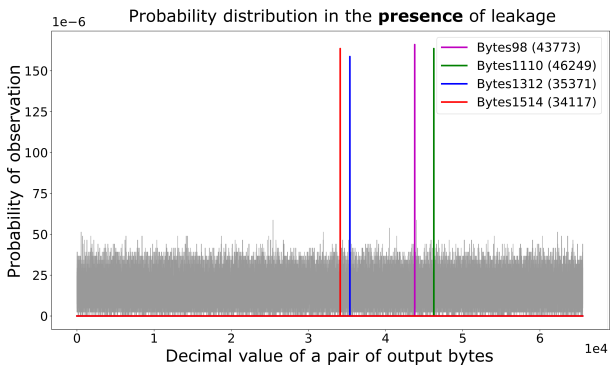


(b) Most significant bytes

Fig. 5: Probability distribution of a pair of ciphertext bytes, for an attacker of 2k ROs on the left side of the FPGA. No secret key leakage can be observed.



(a) Least significant bytes



(b) Most significant bytes

Fig. 6: Probability distribution of a pair of ciphertext bytes, for an attacker of 19k ROs on the left side of the FPGA. The highest peaks correspond to the secret key, confirming the success of the attack.

of faults, the expected ciphertexts are different and yet the HD is ≈30. In the first experiment (in red), an attacker of 2k ROs is placed on the left half of the FPGA; given its small size, no faults are generated and the HD is as expected: oscillating around 30. However, in the second experiment (in blue), an attacker of 19k ROs is managing to cause faults, which culminate with the key leakage in 0.222% of the encryptions, indicated by the Hamming distance falling to zero.

For a pseudorandom sequence of plaintexts at the input, no ciphertext should repeat often. This may allow for the recovery of the key by observing the ciphertexts that repeat at the output [15]. Yet, a relatively large number of samples is required, to rule out all the faulty ciphertexts. Hence, we analyze the data from 100 attack runs. The probability distribution of the consecutive pairs of ciphertext bytes (where byte 15 is the most significant byte), for the weak attacker of 2k ROs is plotted in Fig. 5. Most of the values appear with the probability close to uniform (15×10^{-6}, equivalent to 2^{-16}), with some reaching 60×10^{-6}, approximately. In comparison, Fig. 6 shows the probability distribution of the same pairs of bytes, for the attacker of 19k ROs. In this case, the attack is successful: the values corresponding to the highest eight peaks (reaching above 120×10^{-6}, an order of magnitude higher value than that of the uniform probability distribution), correspond precisely to the secret key. It is worth noting that it was not possible to *visually* identify the secret key from the probability distribution of a single ciphertext byte because the Trojan is activated less frequently under the X-attack than under overclocking [15]. Unlike the variation in the clock frequency, the voltage fluctuations can cause the FPGA to reset and, therefore, need to be carefully controlled. This results in the transient effects of the attack leading to fewer activations of the Trojan.

3) Attacker Size and Placement Analysis: As a step towards better understanding of the mechanism of the attack and the protection against it, we vary the number of ROs for two design floorplans shown in Fig. 7. In the first, the attacker is constrained to the region left of the victim, while in the second, it is constrained to the region above the victim. This time too, Trojan-2 is used. We start the experiments with the smallest possible attacker, to gradually increase its size to 22k ROs (34.3% of the total number of the ALMs). The don't-care signal delays are 3.274 ns and 4.376 ns for the placement in Fig 7a, and 2.639 ns and 4.17 ns for the placement in Fig. 7b.

Fig. 8 shows the Hamming distance between the obtained and the expected victim output. A sequence of only two alternating plaintexts is supplied to the AES; they are chosen so that the don't-care signals change value every clock cycle. The attacker size at which the HD becomes higher then zero

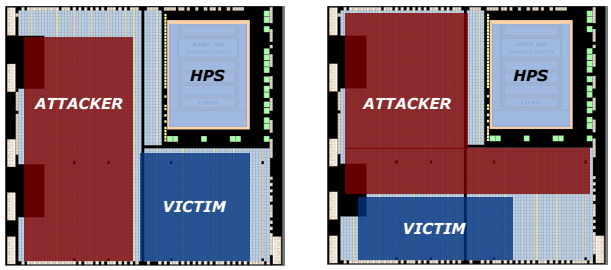


Fig. 7: Tested floorplans. (a) ROs on the left and AES on the right. (b) ROs on the top and AES on the bottom.

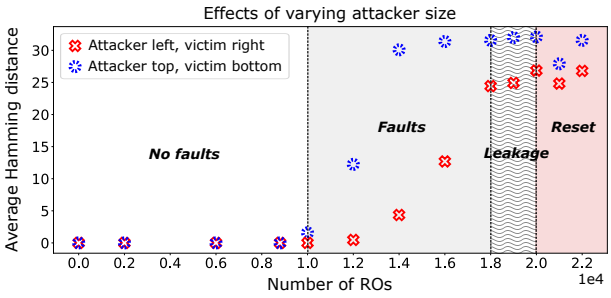


Fig. 8: Hamming distance between the obtained and the expected ciphertexts (for the floorplans in Fig. 7), averaged over 100 attack runs. Vertical dashed lines mark the beginning of appearance of faults, key leakage, and reset.

marks the beginning of appearance of timing faults. To find the attacker size and position that result in the leakage of the secret key, we repeat the attack experiments and the probability distribution analyses described in Section VI-A2.

Consistently with the related work on don't-care SDC Trojans [15], we observe that faults start occurring before the key leakage. This effect is similar to increasing the clock frequency enough for the high-criticality paths to start failing but not enough for the paths of the don't-care signals to fail. With further increase in the number of ROs, the Hamming distance grows. This result, again, corresponds to the effect that even higher clock frequency would cause. And, at 18k ROs, while some of the signal paths are still failing, the Trojan is finally triggered. Augmenting the number of ROs beyond 20k sends the board to reset before the end of the 100 runs of the attack. Finally, we can observe that the main factor of success is the attacker size, i.e., the number of ROs; while the two placements produce slightly different Hamming distance values, the key leakage and the reset occur for equal attacker size.

B. Masked AES S-box Experiments

For an SDC Trojan to be inserted by a malicious user, all that is required is to find a suitable pair of don't-care signals that satisfy the criteria of having only one impossible combination of values. Once this is achieved, the underlying circuit becomes vulnerable to X-attack. Commonly, cryptographic circuits are protected against side-channel attacks [26], but those countermeasures are, in fact, ineffective against fault attacks. To highlight the importance of having a protection against X-attack, we here demonstrate a successful key retrieval from a masked cryptographic core.

As target, we choose an openly-available masked AES S-box [24], [27]. The critical path of the design being 9.58 ns, we set the clock frequency to 100 MHz. Using Yosys [25], we identify the don't-care signal pairs and choose one consisting of a bit of the factor with bit sum and a bit inside the Galois field multiplier. However, the delays of these signals are far from the target clock period (3.8 ns and 3.5 ns, respectively). To increase their propagation time and yet satisfy the trigger criteria, we design the third Trojan variant (Trojan-3 in Section VI-A1) by applying the following signal transformations:

$$\begin{aligned} t_1 &= s_1 \cdot s_2 + x \cdot y, \\ t_2 &= s_1 \cdot s_2 + \overline{x} \cdot y. \end{aligned} \tag{1}$$

Here, s_1 and s_2 correspond to the original don't-care signal pair, y is an arbitrarily chosen S-box signal, while x is a logical function of several S-box signals. Expressions in (1) guarantee that the resulting pair of signals (t_1, t_2) satisfies the Trojan trigger constraints: if $y = 1$, $t_1 = x$ and $t_2 = \bar{x}$; otherwise, $t_1 = t_2 = 0$. Applying these transformations results in considerably slower trigger signals: approximately 8.2 ns.

We test the attack using the floorplan in Fig. 7a, while supplying the victim with pseudorandom plaintexts and masks. As expected, our attack proves to be successful. Experiments with varying attacker size reveal that 17k ROs (27.2% of available ALMs) are sufficient to successfully retrieve the secret key from the masked S-box.

VII. Countermeasures

A. Background and Related Work

Countermeasures against X-attack fall into two categories: Trojan detection and fault attack prevention. Unfortunately, detecting don't-care Trojans can be difficult [15], as Table I suggests. However, detecting and preventing the timing-fault attack should be feasible. For example, by implementing multiple instances of security critical functions and by comparing their outputs, one could detect a fault whenever these redundant results are inconsistent. Lowering the clock frequency and increasing the circuit timing margin is another, albeit not very reliable, alternative. Yet, both of these approaches incur considerable area and performance overheads.

A less resource-consuming alternative could consist in adding a delay (i.e., voltage) sensor to the victim, allowing it to monitor local delay and voltage variations, with the goal of using them to tell that a timing fault *might* have happened. However, the important issues here are that the victim may be unable to reliably calibrate the sensor or to accurately evaluate its steady state value in a shared-FPGA threat model, where tenants not only may have very variable power profiles but may join or leave the FPGA at any point in time. So obtained inaccurate sensor baseline would result in a faulty protection.

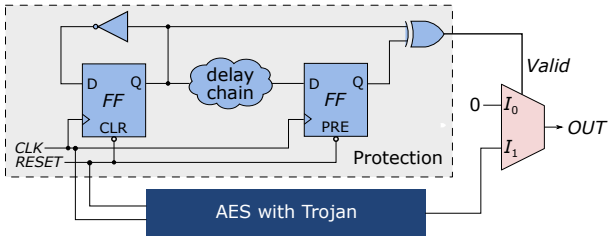


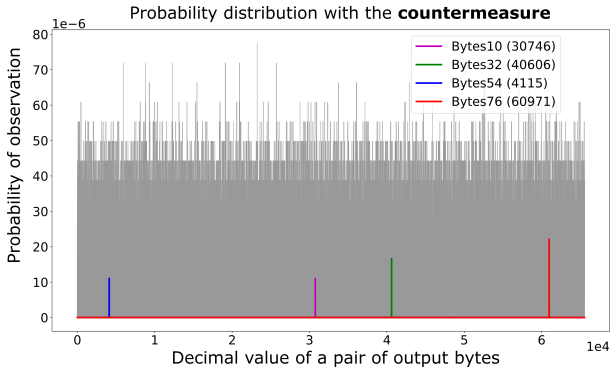
Fig. 9: Our design of a protection against X-attack. In normal operation, *Valid* is set and the AES output is routed to the *OUT* port. However, under the attack, the signal passing through the delay chain suffers from a timing fault; this results in clearing the *Valid* signal, thus forcing *OUT* to constant zero.

Other previously proposed countermeasures include glitch detectors [28], aging sensors [29], [30], shadow registers [31], and razor latches [32]. Assuming that all one knows about the victim is its critical path delay, we design a lightweight countermeasure illustrated in Fig. 9 and place it on the side of the victim. Our design is different from previous in being independent from the protected circuit and not requiring an additional delayed clock in the system. Furthermore, once the attack is detected, we do not attempt to correct the ciphertexts. Instead, we disconnect the IP core from the system output and send out an alarm signal, effectively preventing the leaked secret from being observed.

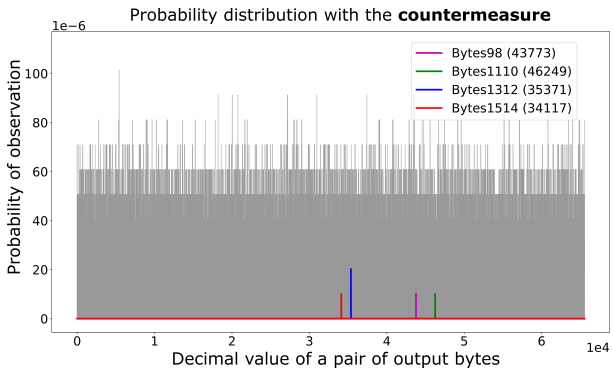
B. Countermeasure Design and Test

The main component in the protection circuit is a delay chain (a sequence of buffers) calibrated to match the longest path delay of the victim. During normal operation, the two registers in Fig. 9 generate opposite signals and the *Valid* signal is set. Consequently, the victim output is routed to the *OUT* port. However, under a strong attack, the signal passing through the delay chain is the first to suffer from a timing fault; as a result, the *Valid* signal is cleared and the *OUT* port receives zeros (or any other *dummy* value one may choose). The circuit in Fig. 9 is more likely to fault during the attack than the data-dependent trigger paths of the hidden Trojan, in particular when they do not lie on the design critical path. With respect to the pipelined AES [23], the protection circuitry occupies 76 ALMs, i.e., only 2.84% of the number of ALMs used by the AES core. Moreover, the protection being placed on the side of the victim, its effect on the maximum design clock frequency is minimized; in our experiments, the design clock frequency remains unchanged.

After confirming that in the absence of the attack and when the attack is weak (configuration of Fig. 5) the protected victim is working as expected, we run the attack setup of Fig. 6. To simulate the worst-case scenario with the SDC signals on the critical path of the AES, we insert a Trojan-3 into the victim. Then, we test several protection circuits with the corresponding delay chains in the range from 0.3 ns below to 1.5 ns above the AES critical path delay. In all the experiments, no key leakage is observed, although the faulty ciphertexts occasionally manage to escape to the output. As expected, the longer the delay chain, the earlier it is activated and the fewer faulty ciphertexts propagate out.



(a) Least significant bytes



(b) Most significant bytes

Fig. 10: Probability distribution of a pair of *OUT* bytes, for the attack setting used in Fig. 6 and the protection logic in place. The delay chain consists of 15 buffers, with an estimated total delay of 0.137 ns above the critical path of the AES core, which is still within the critical path of the entire victim design. Being several orders of magnitude higher than all other values, the probability of zero at the output is not shown. No secret key leakage can be observed, confirming the effectiveness of the proposed protection circuit.

The resulting probability distribution of the pairs of output bytes is shown in Fig. 10. Compared to Fig. 6, the probability of observing the secret at the victim output is now considerably lower. In fact, it is no longer possible to extract the key even by looking for repeated ciphertexts, because the key does not appear at the output. Therefore, our countermeasure is both resource-efficient as well as successful in protecting against the X-attack.

VIII. Conclusion and Future Work

In this work, we presented X-attack, a new attack scenario on shared FPGAs. Our attack induces fluctuations in the on-chip signal delays to remotely activate a dormant satisfiability don't-care hardware Trojan deployed in an isolated private design region. This hard-to-detect Trojan can be inserted into

many designs, due to the common existence of satisfiability don't-cares, and is able to survive from logic synthesis optimizations. We experimentally proved that our attack is effective against both unprotected and well-protected cryptographic circuits. We also investigated attacker configurations that successfully leak the circuit's secret. Finally, we proposed and implemented a lightweight countermeasure, which invalidates circuit outputs produced under the attack. Future work will focus on porting our attack to a cloud environment, using different implementations of power-wasting circuits, and further developing effective defenses that will allow for a more secure FPGA multitenancy.

REFERENCES

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

[2] *Project Catapult*, Microsoft Research, 2019. [Online]. Available: www.microsoft.com/en-us/research/project/project-catapult/

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

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

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

[6] D. Mahmoud and M. Stojilović, "Timing violation induced faults in multi-tenant FPGAs," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE)*, Florence, Italy, Mar. 2019, pp. 1745–50.

[7] O. Glamočanin, L. Coulon, F. Regazzoni, and M. Stojilović, "Are cloud FPGAs really vulnerable to power analysis attacks?" in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE)*, Grenoble, France, Mar. 2020, pp. 1–4.

[8] I. Giechaskiel, K. B. Rasmussen, and K. Eguro, "Leaky wires: Information leakage and covert communication between FPGA long wires," in *Proceedings of 13th ACM ASIA Conference on Information, Computer and Communications Security (ASIACCS)*, Songdo, Incheon, Republic of Korea, Jun. 2018, pp. 15–27.

[9] M. Zhao and G. E. Suh, "FPGA-based remote power side-channel attacks," in *Proceedings of IEEE Symposium on Security and Privacy*, San Francisco, CA, USA, May 2018, pp. 805–20.

[10] 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 *Proceedings of the 26th IEEE Symposium on Field-Programmable Custom Computing Machines*, Boulder, CO, USA, May 2018, pp. 1–8.

[11] F. Schellenberg, D. R. Gnad, A. Moradi, and M. B. Tahoori, "An inside job: Remote power analysis attacks on FPGAs," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE)*, Dresden, Germany, Mar. 2018, pp. 1111–16.

[12] N. Fern, S. Kulkarni, and K.-T. T. Cheng, "Hardware Trojans hidden in RTL don't cares - Automated insertion and prevention methodologies," in *2015 IEEE International Test Conference (ITC)*, Anaheim, CA, USA, Oct. 2015, pp. 1–8.

[13] A. Nahiyan, K. Xiao, K. Yang, Y. Jin, D. Forte, and M. Tehranipoor, "AVFSM: A framework for identifying and mitigating vulnerabilities in FSMs," in *Proceedings of the 53rd Design Automation Conference*, Austin, TX, USA, Jun. 2016, pp. 89:1–6.

[14] C. Krieg, C. Wolf, A. Jantsch, and T. Zseby, "Toggle MUX: How X-optimism can lead to malicious hardware," in *Proceedings of the 54th Design Automation Conference*, Austin, TX, USA, Jun. 2017, pp. 1–6.

[15] 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 *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov. 2017, pp. 707–13.

[16] W. Hu, Y. Ma, X. Wang, and X. Wang, "Leveraging unspecified functionality in obfuscated hardware for Trojan and fault attacks," in *Proceedings of the 2019 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, Xi'an, China, Dec. 2019, pp. 1–6.

[17] K. Matas, T. La, N. Grunchevski, K. Pham, and D. Koch, "Invited tutorial: FPGA hardware security for datacenters and beyond," in *Proceedings of the 28th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, Seaside, CA, USA, Feb. 2020, pp. 11–20.

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

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

[20] Intel, "Development and education boards," 2020. [Online]. Available: software.intel.com/en-us/fpga-academic/teach

[21] K. M. Zick, M. Srivastav, W. Zhang, and M. French, "Sensing nanosecond-scale voltage attacks and natural transients in FPGAs," in *Proceedings of the 21st ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, Monterey, CA, USA, Feb. 2013, pp. 101–4.

[22] I. Giechaskiel, K. B. Rasmussen, and J. Szefer, "Measuring long wire leakage with ring oscillators in cloud FPGAs," in *Proceedings of the 29th International Conference on Field-Programmable Logic and Applications*, Barcelona, Spain, Sep. 2019, pp. 45–50.

[23] H. Hsing, "Tiny AES." [Online]. Available: opencores.org/projects/tiny_aes

[24] D. Canright, "Masked S-box implementation (Verilog)," Nov. 2008. [Online]. Available: faculty.nps.edu/drcanrig/pub/sboxmaskcorr.txt

[25] C. Wolf, "Yosys Open SYnthesis Suite." [Online]. Available: www.clifford.at/yosys/

[26] T. Güneysu and A. Moradi, "Generic side-channel countermeasures for reconfigurable devices," in *Conference on Cryptographic Hardware and Embedded Systems (CHES)*, ser. Lecture Notes in Computer Science (LNCS). Berlin, Heidelberg: Springer, 2011, vol. 6917, pp. 33–48.

[27] D. Canright and L. Batina, "A very compact "perfectly masked" S-box for AES (corrected)," *Cryptology ePrint Archive*, vol. 2009, pp. 1–11, Jan. 2009.

[28] L. Zussa, A. Dehbaoui, K. Tobich, J. Dutertre, P. Maurine, L. Guillaume-Sage, J. Clédière, and A. Tria, "Efficiency of a glitch detector against electromagnetic fault injection," in *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, Mar. 2014, pp. 1–6.

[29] 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 Transactions on Computers*, vol. 66, no. 5, pp. 919–26, 2017.

[30] A. Amouri and M. Tahoori, "A low-cost sensor for aging and late transitions detection in modern FPGAs," in *2011 21st International Conference on Field Programmable Logic and Applications*, Sep. 2011, pp. 329–35.

[31] E. Stott, J. M. Levine, P. Y. K. Cheung, and N. Kapre, "Timing fault detection in FPGA-based circuits," in *2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines*, May 2014, pp. 96–99.

[32] D. Ernst, Nam Sung Kim, S. Das, S. Pant, R. Rao, Toan Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, "Razor: a low-power pipeline based on circuit-level timing speculation," in *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36.*, San Diego, CA, USA, Dec. 2003, pp. 7–18.