2022 25th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS) | 978-1-6654-9431-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/DDECS54261.2022.9770154

Exploiting PUF Variation to Detect Fault Injection Attacks

Troya Köylü, Luiza Garaffa, Cezar Reinbrecht, Mahdi Zahedi, Said Hamdioui, Mottaqiallah Taouil
Computer Engineering Group
Delft University of Technology
Delft, the Netherlands
{T.C.Koylu, M.Taouil}@tudelft.nl

Abstract—**The massive deployment of Internet of Things (IoT) devices makes them vulnerable against physical tampering attacks, such as fault injection. These kind of hardware attacks are very popular as they typically do not require complex equipment or high expertise. Hence, it is important that IoT devices are protected against them. In this work, we present a novel fault injection attack detector with high flexibility and low overhead. Our solution is based on the reuse of a security primitive used in many IoT devices, i.e., ring oscillator (RO) physically unclonable function (PUF). Our results show that we obtain a high detection effectiveness and no false alarms against most popular fault injection attacks based on voltage and clock manipulations.**

Index Terms—**fault injection detection, PUF, RO, IoT, hardware security**

I. Introduction

In the past, faults were only considered during testing of dies to catch manufacturing defects and for reliability issues caused by single event upsets. Nowadays, adversaries can deliberately inject faults into the system to steal data, subvert the control flow of the execution, or escalate privileges [1]. This threat is critical for devices that are highly deployed in the field, which is for example the case for Internet of Things (IoT) [2]. Hence, these devices are susceptible to different fault injection attacks, including cheap and straightforward techniques like voltage manipulations [3] or glitching the clock [4].

Many countermeasures against fault injection attacks such as voltage and clock glitching have been proposed. We can group these countermeasures in three classes: integrity checkers, shielding, and sensors. The first group of countermeasures uses redundancy mechanisms to validate a fault-free operation. These mechanisms can be added in time (e.g., algorithm-wise [5], instruction-wise [6], and via delay mechanisms [7]) or in space (i.e., extra hardware [8]) [9]. However, these mechanisms result in a high-performance penalty or overhead, which limit their application to the IoT devices with strict resource budgets. The second group of countermeasures uses shielding to protect against electromagnetic and laser-based attacks. There are two types of shields: passive and active. Passive shields contain metal meshes that cover the circuit against attacks [10], whereas active shields monitor the data transfer on the mesh to detect irregularities that arise from attacks [11]. Again, shielding also causes overhead and they are typically limited to detecting a single type of attack. In the final countermeasure group of sensors, there are voltage [12] and clock sensors [13]. Due to the different nature of fault attacks, this typically necessitates multiple sensors against each attack technique. Thus, using ring oscillator (RO) or physically unclonable function (PUF)-based sensors become a viability, as they are usually sensitive to changes from multiple sources. This is investigated in a number of studies, but they (i) consider very limited attack cases [14]–[16], (ii) do not take environmental changes into consideration [14], (iii) do not reuse already installed resources so create significant overhead [14], [17], [18], (iv) generate false alarms [17], and (v) use rare or broken PUFs [15], [16]. Specifically, a sensor that even attains 99.9% protection cannot be considered as secure, as an attacker that has infinite time can find a point of vulnerability. Therefore, special countermeasures for IoT are needed, which are not only effective in a variety of situations and attacks but also are lightweight.

In this work we present a novel fault injection attack detector that can be used against a variety of attacks. Our solution is based on the monitoring of the responses of the PUF, which are already installed in the IoT devices as security primitives. In summary, our main contributions are as follows:

- Proposal of an effective, lightweight, flexible (can be used for many applications), adaptive (against aging and environmental changes), and robust (no single point of vulnerability) fault injection detector based on RO PUF.
- Evaluation of the detector under clock attacks (i.e., clock glitch).
- Evaluation of the detector under voltage attacks (i.e., underfeeding and glitching).

The rest of the paper is organized as follows. Section II explains the methodology. Section III describes the experiments that we use to evaluate the detector's performance. Finally, Section IV concludes the paper.

978-1-6654-9431-1/22/$31.00 ©2022 IEEE

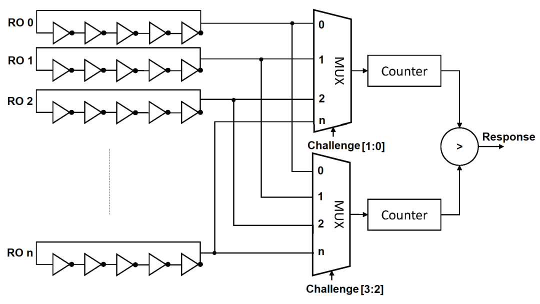

Fig. 1. Example of an RO PUF Architecture [19]

II. Methodology

This section describes our RO PUF-based fault injection detector. Section II-A explains the concept behind our detector. Thereafter, Section II-B describes the working principle of the detector. Finally, Section II-C describes its hardware implementation.

A. Using RO PUFs against Fault Attacks

RO PUFs, whose architecture is shown in Figure 1, create unique responses by comparing different counters. The clock frequency of these counters are determined by the oscillation frequencies of the selected ROs. Each RO is designed with the same number of odd inverters, each equal in size with equal spacing (i.e., they are interconnected exactly in the same way). Hence, in an ideal world, all ROs have the same oscillating frequency. However, due to process variations, the frequency of each inverter slightly differs and as a result, the counters will end up with different values when the amount of clock periods are counted in a certain time window. By comparing the values of the two counters, a single bit response can be generated. Note that more response bits can be generated by using more ROs and more counters. In this PUF, the challenge (i.e., input) defines which ROs are selected to generate the response. For example, in Figure 1, the binary input "0100" selects RO 0 (for the top counter) and RO 1 (for the bottom counter) to generate the response, while input "1000" uses RO 0 (top) and RO 2 (bottom).

In general, the RO PUF is easy to implement, has a medium to low overhead and provides good responses compared to other types of PUFs. However, this PUF is very sensitive to thermal, power supply and noise variations [20] and hence, to fault injection. Such a sensitivity is an important issue in PUFs as they can impact the PUF reliability. To make PUFs reliable, auxiliary hardware is used to correct erroneous bits, like error correction codes [21]. On the other hand, using PUF sensitivity to make sensors [22] in order to monitor the temperature or other environmental conditions have already been proposed. However, using PUFs for sensing fault injection attacks is largely unexplored.

In this study, we assume that the attacker is equipped to perform clock and voltage-based fault injection attacks to leak sensitive information. Hence, the attacker has physical access to an IoT sensor or gateway device that contains an RO PUF for cryptographic operations. This is a very common scenario, as it is the only way to ensure a root of trust in each device when the network is not fully trusted [23]. However, we only assume that the attacker can leak information by injecting faults to sensitive operations and observing the network [24]. Other sophisticated means of leaking information (i.e., side channel analysis) is out of the paper's scope and protection schemes such as masking should be deployed for them [25].

To sense when an attack as defined occurs, we use the embedded RO PUF, as Figure 2 shows at a high-level. Before starting any security sensitive operation, the system activates the detector (indicated by trigger - T on the figure). During the operation, the detector measures and saves the responses of an RO-based PUF (save - S). When the operation ends, the detector compares the collected responses to a reference value, which is pre-collected under similar but no attack conditions (COMPARE). If these values are not identical, an alarm is raised (DECIDE). Consequently, the system can halt the operation, redo the encryption, prevent the output from reaching the user, or provide a random output value (RELEASE OR DROP).

B. Working Principle

The RO PUF-based detector operates in two phases: response collection and decision. First, the response collection phase consists of applying a specific challenge to the PUF multiple times and process each response. The detector keeps collecting responses until the sensitive operation (such as encryption) is completed. At the end of this phase, the detector collects M responses (r) of N bits, where N is given by the number of comparators (see Figure 1). Next, in the decision phase, all M responses are reduced to a single r by using majority voting on each bit of the responses and the resulting r is compared to a reference value r_{ref}. An alarm is raised when $r \neq r_{ref}$.

In order to perform these functions, our PUF must satisfy certain properties. First, it should be able to detect glitches in a short time window. Second, it should be aging and change resistant. Third, it should be integrated into a complete system. These three properties are explained next.

Detecting glitches: To guarantee a high detection efficiency, we require a sensitive PUF, where the response is evaluated after each cycle. Such a PUF would not be useful as a security primitive for the system. To overcome this trade-off, we propose a minor modification to the PUF and have two modes of operation: reliable and unreliable. The difference between the two modes is the number of operation clock cycles used to evaluate the

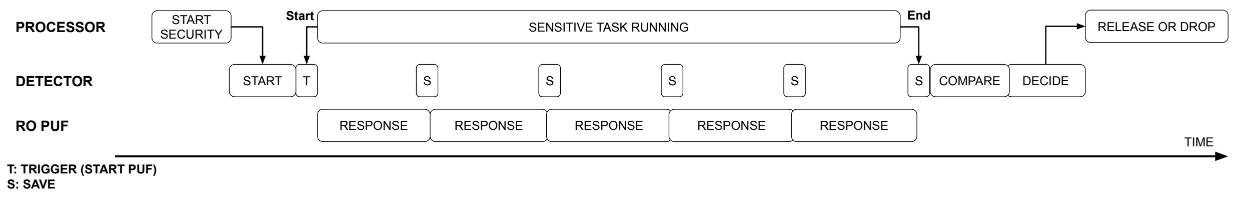

Fig. 2. RO PUF-based Detector Concept

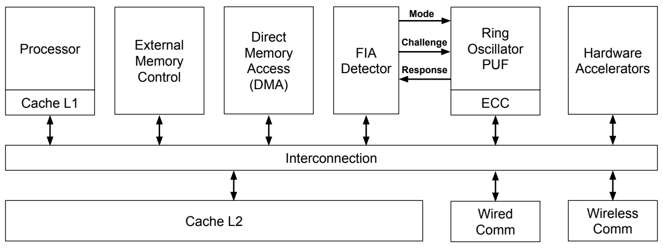

Fig. 3. System-on-chip with the RO PUF based detector

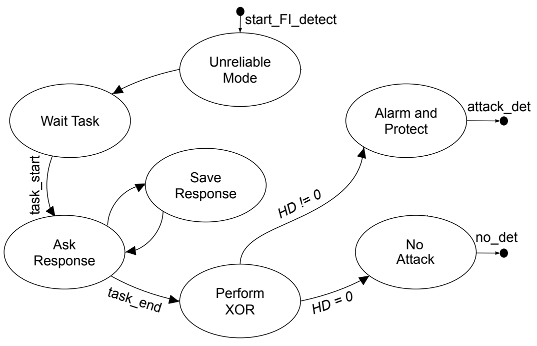

Fig. 4. FSM Implementation of Our Detector

ROs. During reliable operation, the detector produces one response to a challenge after e.g., 100 cycles. This makes the response more stable and hence can be used as a security primitive. During unreliable operation, the PUF produces responses in each clock cycle. These responses naturally differ from the expected value but allow the capture of clock or voltage irregularities.

Aging and environmental change resistant: An in-field PUF ages as it is used, which affects its reliability. This might change the challenge-response pair behavior over time [26]. These changes should not be considered as a fault injection attack. A similar behavior can be observed when the device moves from a warmer to a colder place or vice versa. To address these issues, we propose periodic adjustment of the reference challenge-response pair (c_{ref}-r_{ref}) used by the detector. This can be accomplished by operating the PUF in the reliable mode. For example, by challenging the PUF multiple times, the new reference can be determined by taking the most occurring value or average of the responses. We assume that there are no fault attacks during this operation.

System Integration: Figure 3 shows the SoC architecture in which the proposed fault injection attack (FIA) detector is an IP block. As observed in the figure, the detector is integrated in the same manner as other IPs. The processor can communicate with all the IPs and the detector using the interconnection infrastructure. The fault injection attack (FIA) detector communicates with the PUF, i.e., the detector sets the operation mode of the PUF (i.e., reliable or unreliable) through the *Mode* signal, provides the *Challenge*, and receives/observes the *Response*. Note that the error correction code (ECC) block attached to the PUF is only used in the reliable mode, e.g., for key generation or authentication.

C. Implementation

Figure 4 shows the finite state machine (FSM) implementation of the detector in hardware, for the unreliable mode. Each time the system is about to run a sensitive operation, it triggers the detector through the *start_FI_detect* signal. This signal initiates the FSM with its first state, which forces the PUF into the *Unreliable Mode* state. The FSM subsequently proceeds to its second state, i.e., *Wait Task*, where it waits for the sensitive operation to start. The start of the sensitive operation is indicated by the *task_start* signal. When the operation starts, the FSM alternates between the *Ask Response* state, where it waits for PUF responses; and the *Save Response* state, where the results of the responses are saved.

When the sensitive operation ends, the *task_end* signal notifies the FSM to enter into the comparison state called *Perform XOR*. In this state, the detector performs the majority voting between responses and compares the result with the reference PUF value using XOR operations. In case the numbers are equal (i.e., Hamming distance (HD) equals 0), no attack has taken place. This is signaled in the *No Attack* state. On the other hand, if the XOR results in a non-zero value (i.e., HD is nonzero), the FSM transits into the *Alarm and Protect* state to notify that an attack has taken place. In this state, the detector sets the *attack_det* signal to inform the processor about the attack. The CPU can for example prevent the results from being transmitted to the user.

III. Experimental Results

In this section, we present the experimental setup, clock and voltage glitching/underfeeding experiments, and the obtained results.

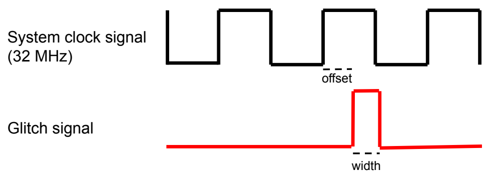

Fig. 5. Clock Glitching

A. Setup

The performance of the detector is evaluated by running experiments on the Chipwhisperer CW305 Artix FPGA Target [27]. We complemented the CW305 board with a CW1173 [28] that acts as a manager, i.e., it initiates the operation, controls the glitching, and collects the results. The FPGA is programmed with a bitstream that contains the design in Figure 3. It runs the AES-128 as hardware accelerator. The PUF contains a single specific challenge with an 8-bit response using eight ROs consisting of three inverting gates (note that some ROs are used in multiple responses). During an encryption, the detector collects four PUF responses. Next, we describe our clock/voltage glitching and voltage underfeeding experiments and their results.

B. Clock Experiments

In the clock experiments, we investigate the detector's fault detection performance during different clock glitching configurations, where 50 AES encryption runs are evaluated per scenario. A scenario is specified by the glitch type. In terms of clock glitches, this is characterized by a glitch width (between -50% and 50% of the clock period) and offset (between -50% and 50%), as illustrated in Figure 5. In our experiments, we used a small part of the glitching range to reduce the number of crashes and make the detection conditions less favorable, while still being able to create effective glitches. For each scenario, we investigate the attack effectiveness (i.e., ratio of the cases that create a faulty output) and report the corresponding detector effectiveness (i.e., ratio of the attacks detected). Table I presents the results. The first two columns specify the configuration of the clock glitch, the third column the attack efficiency (i.e, how many encryptions lead to a corrupt output), and the last column the number of times the detector raised the attack detection flag in percentage. Note that the attack effectiveness does not only consider successful attacks, i.e., attacks that reveal (parts of) the key, but also consider any faulty output in general. The reason for this is that we want to detect how good the detector in general is when an attacker tries to perform fault injection, as our detector can be applied in any sensitive operation.

As can be observed from Table I, our detector is effective in correctly labeling clock glitching scenarios. In only some of the glitch configurations, the AES output became faulty. In all these cases, our detector was able to partially or fully detect these glitches. The detector was even able to detect some cases where the attacks were ineffective. The average detection rate in effective attack scenarios is around 70%. The lowest detection rate is 30%, which is indeed far from preventing most of the attacks for that scenario. We further discuss how to remove such singular points of failure in the next subsection.

TABLE I
EVALUATION RESULTS OF THE CLOCK GLITCHING ATTACK

Clock Glitch		Attack	Detector
Width	Offset	Effectiveness	Effectiveness
1.95	-5	100%	70%
2.73	-5	0%	0%
3.5	-5	0%	0%
4.5	-5	0%	0%
1.95	-3	0%	0%
2.73	-3	0%	0%
3.5	-3	0%	0%
4.5	-3	94%	30%
1.95	1	0%	0%
2.73	1	100%	40%
3.5	1	8%	100%
4.5	1	0%	100%
1.95	3	6%	80%
2.73	3	0%	60%
3.5	3	0%	0%
4.5	3	0%	0%
1.95	5	4%	80%
2.73	5	0%	90%
3.5	5	0%	70%
4.5	5	0%	0%

TABLE II
EVALUATION RESULTS OF THE VOLTAGE UNDERFEEDING

Voltage Underfeeding	Attack Effectiveness	Detector Effectiveness
1.1	0%	100%
1.0 (nominal)	0%	0%
0.85	0%	0%
0.75	0%	0%
0.7	100%	100%
0.65	100%	100%

C. Voltage Experiments

We performed two types of experiments with respect to voltage attacks. The first one is voltage underfeeding. In this attack, the attacker supplies a voltage outside the nominal range to the device, where 1V is the nominal value and the voltage range 1.1 - 0.9V is considered to be the optimal operating condition. Table II presents the results for different voltages. The table is constructed in a similar manner as Table I.

The first important discussion from Table II is related to the voltage values 1.1, 1.0, and 0.85V. As mentioned before, the first two voltages fall in the optimal condition range, and the last one under normal condition. When supply voltages of 1V and 0.85V are applied, we observed that our detector does not raise any false alarms. The detector does raise alarms when a 1.1V supply voltage is used. However, note that we configured our detector solely on the nominal voltage and hence, the detector is able to detect this voltage setting. Therefore, it is not straightforward to label 1.1V cases as false alarms. In order to prevent them, the detector should be charac-

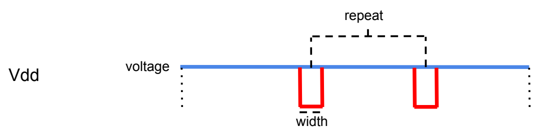

Fig. 6. Voltage Glitching

TABLE III
EVALUATION RESULTS OF THE VOLTAGE GLITCHING EXPERIMENTS

Voltage Glitch			Attack	Detector
Voltage	Width	Repeat	Effectiveness	Effectiveness
1.0	3.5	1	0%	0%
0.85	3.5	1	0%	0%
0.75	3.5	1	0%	0%
1.0	3.5	4	0%	0%
0.85	3.5	4	100%	60%
0.75	3.5	4	100%	100%
1.0	3.5	10	100%	0%
0.85	3.5	10	100%	100%
0.75	3.5	10	100%	100%
1.0	45	1	0%	0%
0.85	45	1	0%	0%
0.75	45	1	0%	0%
1.0	45	4	0%	0%
0.85	45	4	100%	50%
0.75	45	4	100%	100%
1.0	45	10	100%	0%
0.85	45	10	100%	100%
0.75	45	10	100%	100%

terized and verified based on this voltage setting as well (i.e., change of reference value, see Subsection II-B). Second, our detector perfectly detects the successful glitches in the cases where voltage underfeeding took place. Note that both the attack and detector effectiveness are 100% for these cases.

The second set of voltage experiments is related to voltage glitching. Table III presents the results in a similar manner as the previous tables. The voltage glitches are characterized by the glitch width (in percentage) and how often they are repeated, as illustrated in Figure 6. Each time a glitch occurs, the V_{dd} is shorted towards 0V. In the table, the voltage column represents the operating voltage. Only the voltages where the attack effectiveness is 0% in Table II have been considered, as the detector can detect all the other supply voltages with 100% effectiveness.

The table shows that the detector performs well in many scenarios. Overall, the detection effectiveness for effective attacks is again around 70%. However, in 2 cases some effective attacks are not detected by the detector. These occur only at nominal supply voltage.

For a more in-depth analysis, we analyzed the PUF responses in each of the experiments. In this analysis, we observed that a wrong response for a scenario randomly alternates between a specific set of values. Figure 7 presents the plot of obtained PUF responses for all three experiments: clock glitching, voltage underfeeding, and voltage glitching. The fault-free reference response (r_{ref}) is 44, indicated by the white bars in the figure.

The plot shows that some responses are close to the reference value of 44, while some are very distant. The larger the difference with the reference value, the more

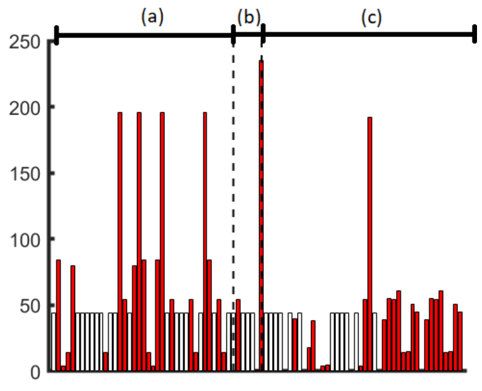

Fig. 7. Unique PUF Response for attack scenario (a) clock glitching, (b) voltage underfeeding, and (c) voltage glitching.

likely that the attack causes bit-flips in the design. It can be observed that the cases with a larger difference are in greater proportions in clock glitching and voltage underfeeding attacks. For the voltage glitching case, there is a greater number of faulty responses closer to the reference. As noted previously, some effective voltage attack scenarios indeed managed to escape our detection.

The undetected cases can be an issue, especially when attackers are able to perform various glitching experiments to discover these voltage glitch values [29] (i.e., voltage, glitch width, and repeat values - see Table III). However, this can be improved in several ways. First, instead of looking at majority voting of the responses, each individual PUF response can be analyzed and compared to the reference. Second, aperiodic changing of the inverter chain length can alter the PUF sensitivity at run-time; this increases the detection probability of currently undetected cases.

We made another set of experiments to validate these two proposed improvements. Our experiments show that the individual PUF responses show much more variance when glitches occur (mean μ=63.27, standard deviation σ=83.81) as compared to the case when no fault injection takes place (μ=33.77, σ=61.06). Second, when we used five inverters instead of three, we observed a different distribution (i.e., with glitching μ=35.02 and σ=60.42 and without glitching μ=17.33 and σ=36.31), which indeed impacts the PUF sensitivity. This however must be used carefully, as this might comprise the reliability/reproducability of the PUF function when used as security primitive.

One point we did not discuss yet is the attacks against our detector itself. Our detector is generally robust against them, i.e., any such attack would destabilize the PUF response, resulting in an unexpected behavior. One point of weakness however is the reference response. If an attacker is able to change this value, the system will start raising a lot of false alarms. This is not directly a security problem, but an attacker can deny the operation of the device in this manner. Hence, selective hardening of the reference response should be considered in cases where denial of service must be avoided.

D. Overhead

As mentioned in Subsection III-A, we implemented our PUF-based detector on Chipwhisperer CW305 board. The RO PUF and detector require 53 LUTs and 16 registers, compared to 2506 LUTs and 980 registers required for the interface, hardware AES core, and a couple of 8-bit registers to save the PUF responses. The implementation does not include the response comparison as it can be carried out by the software, but with using XORs, the added overhead is minimal. This shows a very low overhead, especially when the PUF would be reused for authentication purposes. In that particular case, a single challenge can be used in the reliability mode (see Subsection II-B). Moreover, the cost for saving the responses can also be further reduced by comparing them on the fly as they are produced. Note that the design satisfies all timing constraints and that the overhead is smaller or comparable to the state of the art.

IV. Conclusion

This paper presented a novel RO PUF-based fault injection attack detector. Experimental results show that our low-cost detector is effective against many cases of clock and voltage-based attacks. Our RO PUF-based detector is one of the early designs where built-in PUFs are re-purposed against various fault injection attacks. Furthermore, it is robust against changing environmental conditions and aging, and provides resistance to a single point of vulnerability. As RO PUFs are sensitive to temperature, EM, and lasers [17], [30], [31], our method has the potential to be used against these attacks as well. Although we only tested our detector with a hardware AES, our method is general and can be used with any security sensitive operation.

V. Acknowledgment

This work was labelled by the EUREKA cluster PENTA and funded by Dutch authorities under grant agreement PENTA-2018e-17004-SunRISE.

References

[1] J. Mickens. (2020) Software-level attacks on architectural and microarchitectural state. HiPEAC. [Online]. Available: https://www.hipeac.net/media/private/files/73/10/ACACES-2020-part-I.pdf

[2] W. Kassab *et al.*, "A–z survey of internet of things: Architectures, protocols, applications, recent advances, future directions and recommendations," *Journal of Network and Computer Applications*, vol. 163, 2020.

[3] A. Barenghi *et al.*, "Low voltage fault attacks on the rsa cryptosystem," in *FDTC*. IEEE, 2009.

[4] T. Fukunaga *et al.*, "Practical fault attack on a cryptographic lsi with iso/iec 18033-3 block ciphers," in *FDTC*. IEEE, 2009.

[6] A. Barenghi *et al.*, "Countermeasures against fault attacks on software implemented aes: effectiveness and cost," in *Proceedings of the 5th Workshop on Embedded Systems Security*, 2010.

[5] A. Boscher *et al.*, "Fault resistant rsa signatures: Chinese remaindering in both directions." *IACR Cryptol. ePrint Arch.*, vol. 2010, 2010.

[7] L. Anghel *et al.*, "Cost reduction and evaluation of a temporary faults-detecting technique," in *Design, Automation, and Test in Europe*. Springer, 2008.

[8] R. Karri *et al.*, "Fault-based side-channel cryptanalysis tolerant rijndael symmetric block cipher architecture," in *Proceedings 2001 IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*. IEEE, 2001.

[9] T. C. Koylu *et al.*, "Rnn-based detection of fault attacks on rsa," in *ISCAS*, 2020.

[10] H. Bar-El *et al.*, "The sorcerer's apprentice guide to fault attacks," *Proceedings of the IEEE*, vol. 94, 2006.

[11] X. T. Ngo *et al.*, "Cryptographically secure shield for security ips protection," *IEEE Transactions on Computers*, vol. 66, 2016.

[12] K. M. Zick *et al.*, "Sensing nanosecond-scale voltage attacks and natural transients in fpgas," in *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*, 2013.

[13] K. Sun *et al.*, "Fault-tolerant cluster-wise clock synchronization for wireless sensor networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 2, 2005.

[14] C. Deshpande *et al.*, "A configurable and lightweight timing monitor for fault attack detection," in *ISVLSI*. IEEE, 2016.

[15] K. Shimizu *et al.*, "Puf as a sensor," in *GCCE*. IEEE, 2015.

[16] G. Hammouri *et al.*, "Novel puf-based error detection methods in finite state machines," in *International Conference on Information Security and Cryptology*. Springer, 2008.

[17] S. Tajik *et al.*, "Pufmon: Security monitoring of fpgas using physically unclonable functions," in *IOLTS*. IEEE, 2017.

[18] Y. Yao *et al.*, "Programmable ro (pro): A multipurpose countermeasure against side-channel and fault injection attack," *arXiv preprint arXiv:2106.13784*, 2021.

[19] H. Martin *et al.*, "Enhancing puf based challenge–response sets by exploiting various background noise configurations," *Electronics*, vol. 8, 2019.

[20] S. Docking *et al.*, "A method to derive an equation for the oscillation frequency of a ring oscillator," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 50, 2003.

[21] M.-D. Yu *et al.*, "Secure and robust error correction for physical unclonable functions," *IEEE Design & Test of Computers*, vol. 27, 2010.

[22] Y. Gao *et al.*, "Puf sensor: Exploiting puf unreliability for secure wireless sensing," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, 2017.

[23] C.-C. Chang *et al.*, "Signature gateway: Offloading signature generation to iot gateway accelerated by gpu," *IEEE Internet of Things Journal*, vol. 6, 2018.

[24] G. Piret *et al.*, "A differential fault attack technique against spn structures, with application to the aes and khazad," in *International workshop on cryptographic hardware and embedded systems*. Springer, 2003.

[25] A. Althoff *et al.*, "Hiding intermittent information leakage with architectural support for blinking," in *ISCA*. IEEE, 2018.

[26] R. Maes *et al.*, "Countering the effects of silicon aging on sram pufs," in *HOST*. IEEE, 2014.

[27] "Cw305 artix fpga target," 2016. [Online]. Available: https://rtfm.newae.com/Targets/CW305\%20Artix\%20FPGA/

[28] "Cw1173 chipwhisperer-lite," 2015. [Online]. Available: https://rtfm.newae.com/Capture/ChipWhisperer-Lite/

[29] N. Timmers *et al.*, "Controlling pc on arm using fault injection," in *FDTC*. IEEE, 2016.

[30] W. He *et al.*, "Ring oscillator under laser: potential of pll-based countermeasure against laser fault injection," in *FDTC*. IEEE, 2016.

[31] W. He *et al.*, "Cheap and cheerful: A low-cost digital sensor for detecting laser fault injection attacks," in *International Conference on Security, Privacy, and Applied Cryptography Engineering*. Springer, 2016.