Laser Attack Benchmark Suite

Burin Amornpaisannon
National University of Singapore

Andreas Diavastos
National University of Singapore

Li-Shiuan Peh
National University of Singapore

Trevor E. Carlson
National University of Singapore

ABSTRACT

Laser fault injection in integrated circuits is a powerful information leakage technique due to its high precision, timing accuracy and repeatability. Countermeasures to these attacks have been studied extensively. However, with most current design flows, security tests against these attacks can only be realized after chip fabrication. Restarting the complete silicon design cycle in order to address these vulnerabilities is thus both time-consuming and costly. To overcome these limitations, this paper proposes an open-source benchmark suite that allows chip designers to simulate laser attacks, and evaluate the security of their designs, both hardware-based and software-based, against laser fault injection early on during design time. The proposed benchmark suite consists of a tool that automatically integrates hardware-based spatial, temporal and hybrid redundancy techniques into a target design. With the tools used in this work, we demonstrate how the attacks can be simulated on a Verilog simulator, and run on an FPGA with a design equipped with hardware-based redundancy techniques without manual modifications. This work consists of four attacks, and four hardware-based redundancy techniques. The attacks and defenses together that the benchmark suite provides will automate the entire early design evaluation flow against laser fault injection attacks.

KEYWORDS

Hardware security, benchmark suite, integrated circuits, laser fault attack

ACM Reference Format:
Burin Amornpaisannon, Andreas Diavastos, Li-Shiuan Peh, and Trevor E. Carlson. 2020. Laser Attack Benchmark Suite. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD '20), November 2–5, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3400302.3415646

1 INTRODUCTION

Physical attacks are increasingly becoming a major threat due to an exponential increase in connected devices in the Internet of Things era, where attackers can readily gain physical access to devices. Physical attacks are categorized as active and passive attacks [24]. Active attacks, also called fault attacks, utilize equipment to generate, for example, clock glitches, electromagnetic or laser irradiation to introduce faults into a target system, which lead to faulty system behavior. Confidential information can be retrieved by comparing faulty and correct outputs. Passive attacks, also called side-channel attacks, observe electrical properties of a target system such as power consumption and electromagnetic emissions, and deduce confidential information from these channels' variations.

Laser fault injection is one of the most powerful tools for generating active attacks. This is due to its high precision, timing accuracy and repeatability [11] [29]. Cryptographic algorithms proved to be mathematically secure, such as AES, can leak secret keys in the presence of laser irradiation, which allows the attackers to have access to confidential information, by, for example, skipping an instruction or directly injecting faults into intermediate data [10] [32] [33]. Neural networks, increasingly used in safety-critical applications, running on an embedded system have also been shown to be vulnerable to laser attacks, leading to incorrect predictions when the system is being attacked [9].

Clearly, there is a need to avert laser fault injection attacks. However, chip designers typically have to first finalize and then fabricate their designs before they can test them against laser fault injection attacks. If vulnerabilities are found during post-fabrication testing, the chip designers inevitably have to redo the entire design. These steps, from RTL design to actual chip fabrication, are not only costly, but also time-consuming. Hence, traditional post-fabrication security evaluation occurs too late in the design flow [25]. Chip testing for security also involves expensive, highly sophisticated equipment for generating laser fault attacks, and requires skilled technicians to operate, and depackage the chip to be able to generate the attacks [11]. It is thus impractical to evaluate chip security against laser fault injection attacks only after fabrication. A framework that allows circuit designers to evaluate their designs early on in the design flow is critically needed.

In this paper, we propose the Laser fault Attack Benchmark Suite (LABS), an open-source tool that allows circuit designers to evaluate their design at the early RTL stage and on an FPGA against physical laser attacks. LABS is comprised of a laser attacks benchmark suite that aims to accelerate security testing against laser fault injection by enabling circuit designers to inject faults using provided fault models to verify their chips, and evaluate their protection mechanisms, both hardware-based and software-based, early on in the design flow. It aims to be independent of the specific hardware description languages used to implement the design, and is thus realized flexibly in Chiffre [18] based on FIRRTL [20], which is an open-source hardware intermediate representation. Any hardware design languages that can be converted to FIRRTL are compatible

ICCAD '20, November 2–5, 2020, Virtual Event, USA
© 2020 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-8026-3/20/11.
https://doi.org/10.1145/3400302.3415646

with LABS[1]. The attacks can be run on a Verilog simulator and/or an FPGA without manual design modifications.

LABS also consists of an automated methodology to integrate fault tolerant structures into the specific part of the design that needs to be protected, as defined by the hardware-based fault tolerant techniques. Similar to today's compilers, which can automatically add software-based fault tolerant techniques to target code to detect and recover from errors [15], LABS, realized as compiler passes in the FIRRTL hardware compiler framework, can automatically integrate hardware-based fault tolerant techniques into the circuit design, helping circuit designers evaluate their design defense against physical attacks readily without manual modifications.

Together, LABS generates physical laser attacks, and automates the deploying of hardware-based fault tolerance defenses into the design, thus automating the entire early design evaluation flow against laser fault injection attacks. To the best of our knowledge, there exists no prior laser fault attack benchmark suite.

As an illustration of the potential use scenarios of LABS, we present case studies of attacks on software-implemented AES and neural networks running on the Rocket core [2] implemented in Chisel, an AES accelerator [34] implemented in Verilog, and defenses on the AES accelerator in the experimental results section.

2 MOTIVATION

Laser fault injection attacks rely on parasitic currents generated by laser shots [21] that produce undesired transient voltage, propagating through the logic which can potentially invert bits at the inputs of registers [40]. In Table 1, we outline the characteristics of the three basic levels of laser fault injection attack simulation techniques.

A laser attack at the *Physical level* is an active fault injection technique that uses specialized laser probes to induce high-precision faults. However, a physical laser fault injection attack is a complex and costly process. First, the chip has to be decapped, have the passivation layer removed, and the shielding needs to be circumvented using time consuming and labor intensive chemical or mechanical decapsulations [11, 38], risking the chip to be damaged in the process. To achieve realistic accuracy, it requires expensive laser equipment that can match the technology of the processing chip. While each laser probing only takes a few minutes to perform, this technique is only available after the chip is fabricated. It therefore makes it difficult if not infeasible to upgrade the circuit under test with the necessary countermeasures (to tackle the vulnerable parts found) as it would have to go through the costly design and fabrication process that takes months to complete.

At the *Electrical level*, a double exponential current source can model the first order of a laser shot [26]. With the current sources added to the netlists of cells illuminated by the laser, an electrical-level simulation that takes into account the effects of a laser attack can be performed [41]. To improve simulation performance, new multi-level techniques propose hybrid solutions that simulate in detail only specific circuit blocks that are affected by the laser (see Section 8). However, a fast multi-level electrical simulation requires gate-level simulation to simulate the rest of the design. This reduces the performance of the simulation and increases the total testing cycle of a circuit design against laser attacks. More importantly, electrical-level simulation only simulates the effects of a laser attack on a specific circuit block. It cannot provide insight as to what the effects of the attack will be on executing the application that we aim to protect. Current state-of-the-art electrical level simulations require proprietary tools (i.e. Cadence Voltus^{TM} [13] and Cadence Spectre XPS [12]) to perform, increasing the cost of simulation.

Lower-level techniques provide more realistic accuracy in terms of simulation results as they take into account both the layout of the circuit and the parameters of the laser (i.e. wavelength, spot size, pulse width, energy, position and duration). However, *Logical level* fault modelling of a laser fault injection attack allows for fast simulation of selected logic in a larger system at an early stage of the design process that takes into account the application we aim to protect. LABS provides low-cost and fast security validation of chip designs and hardware-based fault-tolerant integration tool. All tools proposed in this work are open-source, while the flexibility of the framework allows the user to implement and integrate their own attack models and countermeasures. In addition, this methodology is technology independent, as it is implemented in high-level source-code, and does not require prior fabrication of the chip or back-end generation of the design layout targeted to a specific process.

[1]Yosys [44], used in our methodology, supports Verilog and SystemVerilog hardware input types and generates FIRRTL.

Table 1: Laser Fault Attack Simulation Techniques.

Simulation	Physical	Electrical	Logical (LABS)
Speed	Slow	Slow	Fast
Fault Model	Realistic	Current Source	User-defined
Cost	High	Medium	None
Availability	Fabrication	Layout	Source-code
Technology	Dependent	Independent	Independent
Risk of Damage	Yes	No	No

3 LASER FAULT INJECTION AND MITIGATION METHODOLOGY

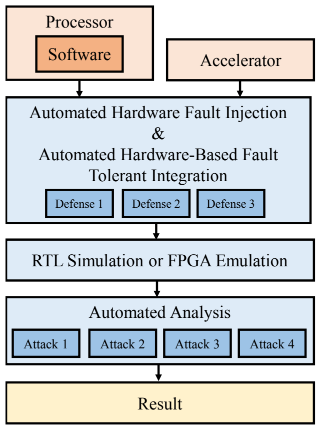

Processor
Software
Accelerator
Automated Hardware Fault Injection
&
Automated Hardware-Based Fault Tolerant Integration
Defense 1
Defense 2
Defense 3
RTL Simulation or FPGA Emulation
Automated Analysis
Attack 1
Attack 2
Attack 3
Attack 4
Result

Figure 1: Overview of LABS. The light blue boxes show the three key parts of LABS consisting of automated hardware fault injection, hardware-based fault tolerant integration and analysis.

Table 2: Attacks in the benchmark suite.

Application	Target	Description
AES	Processor	Skip the last round *Addroundkey* [10]
AES	Processor Accelerator	Inject one-bit fault into input of the last round [19]
RSA−CRT	Processor	Inject faults into one of two parts of signature [8]
Neural Network	Processor	Skip a computation of the activation functions [9]

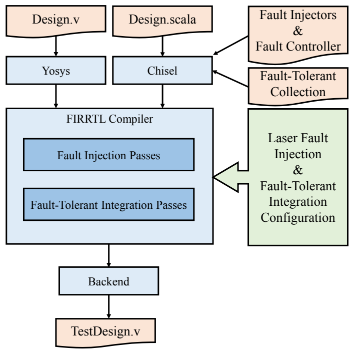
The overview of the flow of the LABS framework comprising the benchmark suite and associated toolchain is shown in Figure 1. The framework supports both processors and accelerators, and thus can be used to test both software-based and hardware-based applications and countermeasures. Table 2 lists the attacks covered in our benchmark suite, AES, RSA−CRT and Neural Networks. AES is a symmetric block cipher, standardized by NIST [28], that has become the global standard for data encryption. RSA−CRT is a fast version of the original RSA algorithm, which is a widely used public key algorithm, based on the Chinese Remainder Theorem. Deep learning has been widely deployed in biometrics applications such as face and voice recognition, and safety-critical applications such as autonomous vehicles, and will be a key part in future smart cities [9]. Attacks on these applications will be catastrophic.

The toolchain consists of three key components: (1) automated hardware fault injection, (2) automated generation and integration of hardware-based fault-tolerant designs and (3) automated fault analysis.

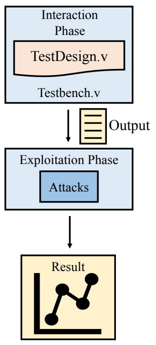
The automated hardware fault injection component is based on Chiffre [18], a configurable hardware fault injection framework, which automatically integrates synthesizable fault injectors and fault controllers into a target design, to inject a specified fault to a target component at a specific time. Chiffre originally only supports bit flips, which do not suffice for laser faults. We thus introduced two additional fault models, bit-set and bit-reset to cover all fault models that can occur by laser fault injection [39]. These faults can be user-defined, generated probabilistically or at random. Our fault controller is added to the framework, and used to observe the target signal, sending an activate signal to all fault injectors when the firing condition is met. For example, the target signal can be an address signal of an instruction being executed, and the fault injectors will be activated when the target instruction is found.

The second component of our toolchain enables automated generation and integration of hardware-based fault-tolerant defenses. For example, to integrate double modular redundancy (DMR), it automatically duplicates the target design, generates modules required for DMR with an appropriate width for the target design, inserts a fault detection port, and connects the modules with the modified design, providing the new design that readily supports the technique. The new design is able to detect errors, and hide its output when errors are detected.

Finally, the third component performs automated fault analysis. It runs one of the attacks shown in Table 2 on the outputs of the design to evaluate the impact of the injected faults. The evaluation result is then generated as a graph, for example, showing how many secret key bytes are revealed, similar to the graphs illustrated later in the experimental results section.


Design.v
Design.scala
Fault Injectors & Fault Controller
Yosys
Chisel
Fault-Tolerant Collection
FIRRTL Compiler
Fault Injection Passes
Fault-Tolerant Integration Passes
Laser Fault Injection & Fault-Tolerant Integration Configuration
Backend
TestDesign.v

(a) LABS's automated hardware fault injection and hardware-based fault-tolerant integration flow.



(b) LABS's simulation and analysis flow.

Figure 2: The end-to-end LABS methodology.

To use LABS, the user first inserts a target design with a configuration. The design can be a processor, which will be running a software application, or a hardware accelerator. The configuration indicates which component will be attacked, and/or hardware-based redundancy technique that will be integrated into a target component in JSON file format shown in Listing 1. Thereafter, the framework generates a synthesizable RTL test design based on the configuration, which can be run using RTL simulation or FPGA emulation. The outputs of the test design are then fed to the analysis component, which runs an attack described in the next section.

```
[{"class":"chiffre.passes.FaultInjectionAnnotation", (a)
  "target":"aes.aes_encipher_block.block_w3_reg",
  "id":"main",
  "injector":"chiffre.inject.FaultInjector" },
 {"class":"chiffre.passes.ScanChainAnnotation",      (b)
  "target":"aes.FaultController.scan",
  "ctrl":"master",
  "dir":"scan",
  "id":"main" },
 {"class":"labs.passes.FaultControllerAnnotation",   (c)
  "target":"aes.aes_encipher_block.round_ctr_reg",
  "data_target":"h_a",
  "max_number_of_fires": 1,
  "target_bits": [1] },
 {"class":"labs.passes.FaultTolerantTMRAnnotation",  (d)
  "target":"aes.aes_encipher_block.None"}]
```

Listing 1: An example of the JSON configuration used by the framework. It consists of four entries: (a) Component to be attacked and its fault fault injector, (b) Fault controller name, (c) Configuration of fault controller, and (d) Insertion of fault-tolerant structures to a target component.

4 LASER FAULT ATTACK BENCHMARKS

Figure 2a shows the flow of the FIRRTL hardware compiler framework that is used in this work. First, a target design implemented in Verilog or Chisel is converted to FIRRTL using Yosys [44], or the Chisel frontend respectively. Note that other hardware design languages that can be converted to FIRRTL can also be used. Then, the compiler passes generate fault controllers used to control fault injectors, and inserts it to the target design. Next, fault injectors are generated, connected to all target components, and connected to the fault controllers. The Verilog RTL test design is then generated from the modified FIRRTL file.

Figure 2b shows the next steps in the simulation of laser attacks, after obtaining a test design. There are two phases for generating physical attacks: the interaction phase, when an attacker tries to attack a circuit physically to obtain desired information, for example, faulty ciphertexts, and the exploitation phase, when the attacker analyzes the information to retrieve confidential information [24]. In LABS, when the test design arrives at the interaction phase, it is simulated with the provided testbench with a Verilog simulator or emulated on FPGA. The testbench collects the outputs of the test design required for the next phase, and outputs a simulation waveform. Next, in the exploitation phase, which is the analysis part in Figure 1 consisting of our suite of laser attack benchmarks implemented in Python, the outputs are fed to analyze the vulnerability of the test design. We detail the four benchmarks below.

AES Attack by Breier et al. [10] Performed experiments in this work show that a micro-controller running an AES algorithm is vulnerable to laser fault injection at the back side of the chip. The attack is described below.

$$C = ShiftRows(SubBytes(M)) \oplus K \tag{1}$$

$$D = ShiftRows(SubBytes(M)) \tag{2}$$

$$K = C \oplus D \tag{3}$$

Let K be the last round key, M be the ninth round temporary ciphertext, C be the correct ciphertext and D be a faulty ciphertext. Laser fault injection is performed to skip the xor instruction used to compute the last round *AddRoundKey*, which makes the faulty ciphertext to be the output of the last *ShiftRows* shown in (2). The last round key can be retrieved with one pair of correct and faulty ciphertexts by xor-ing them (Equation (3)). Thus, after getting the last round key from (3), the actual secret key can then be revealed using the inverse key schedule algorithm. For example, the user can attack a register storing the xor instruction to skip them. The benchmark generates a graph showing how many last round key bytes are needed to be revealed.

AES Attack by Giraud et al. [19] The authors in this work propose an attack on AES that requires a one-bit fault inside intermediate data during the start of the last round. The attack requires around 50 faulty ciphertexts and one correct ciphertext to reveal the entire ninth round temporary ciphertext. The experiments in [17] show that inducing single-bit faults using laser fault injection in the recent 28nm CMOS technology node is still achievable. The attack is described below.

Equation (1) can be written as the equation below where i is a byte number from 0 to 15.

$$C_{ShiftRows(i)} = SubBytes(M_i) \oplus K_{ShiftRows(i)} \tag{4}$$

If there is a one-bit fault e introduced at the byte number j during the beginning of the final round, the result of the faulty output from (4) will be:

$$D_{ShiftRows(j)} = SubBytes(M_j \oplus e_j) \oplus K_{ShiftRows(j)} \tag{5}$$

Note that the one-bit fault will affect only one byte of the output ciphertext. By comparing the correct and faulty ciphertexts, the ninth round intermediate ciphertext at the byte affected can be guessed using the formula below.

$$C_{ShiftRows(j)} \oplus D_{ShiftRows(j)} = SubByte(M_j) \oplus SubByte(M_j + e_j) \tag{6}$$

The left side of Equation (6) is known from the outputs. For the right side, the attacker has to brute force all possible one-bit faults e_j and one-byte temporary ciphertext M_j. All M_j candidates that satisfy (6) will be counted. With several faulty ciphertexts, the correct M_j, that always satisfies the equation, will be counted the most, and thus doing this for every byte will reveal the entire ninth round temporal ciphertext, which can further be used to reveal the secret key. For example, to attack an AES accelerator, the user can inject faults into state registers directly at the start of the last round. The benchmark generates a graph showing how many ninth round temporary ciphertext bytes are left to be revealed.

RSA-CRT Attack by Boneh et al. [8] The RSA-CRT attack, often referred as the Bellcore attack, focuses on the implementation of the RSA public-key algorithm based on the Chinese Remainder Theorem, theoretically showing that software and hardware errors present during the computation of a signature lead to the leakage of a secret exponent using a pair of correct and faulty signatures. The authors in [37] show that the Bellcore attack can be realized, and software countermeasures that have been proposed to protect the algorithm can be bypassed using laser fault injection.

$$S_p = C^d(modp) = C^{dmodp-1}(modp) \tag{7}$$

$$S_q = C^d(modq) = C^{dmodq-1}(modq) \tag{8}$$

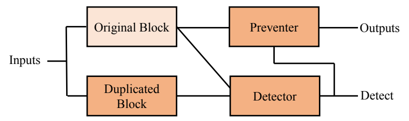
$$S = CRT(S_p, S_q) = S_q + q \cdot ((S_p - S_q) \cdot (q^{-1}modp)modp) \tag{9}$$

Let d be a secret signing exponent. The equations above show how RSA-CRT computes a digital signature S. The faulty signature can be achieved by injecting any faults into S_p in (7) or S_q in (8), not both. Then, the difference between the correct signature S and the faulty signature $\hat{S}$ will leak one of the prime numbers by calculating GCD(S − $\hat{S}$, N), where N is the product of the chosen two prime numbers N = p · q, which is known from the public key. For example, the user can inject a fault into the ALU or skip an instruction during the computation of S_p. The benchmark calculates the outputs using the formula above, and shows a retrieved prime number.

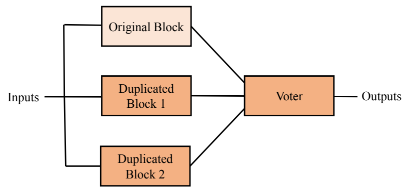
Deep Learning Attack by Breier et al. [9] This work proposes a practical attack that injects faults into neural networks running on an embedded system to skip target instructions used inside an activation function, such as ReLu, sigmoid or tanh, to make predictions of the neural networks incorrect, and shows the first study of using laser fault injection to attack a neural network system.

$$sigmoid(x) = \frac{1}{1 + exp^{-x}} \tag{10}$$

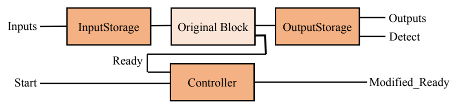
The sigmoid equation is shown in (10). Attackers can make the neural networks predict wrongly by skipping the negation instruction in the exponent function of the sigmoid function. Skipping the negation instruction horizontally flips the graph of the sigmoid function, meaning that the output of the target neuron will be equal to 1 − y, where y is the correct output value. It is suggested that the target layer has to be as close to the output layer as possible to increase misclassification rate. For example, the user can inject faults into the instruction cache to skip the target instruction. The exploitation phase in this attack becomes an analysis phase used to evaluate the accuracy of the neural network, and shows how many samples are correctly classified.



(a) Double Modular Redundancy



(b) Triple Modular Redundancy



(c) Temporal Redundancy

Figure 3: Supported hardware-based redundancy techniques.

5 AUTOMATIC HARDWARE-BASED REDUNDANCY INTEGRATION

LABS' hardware-based redundancy integration tool supports four redundancy techniques: double modular, triple modular, temporal and hybrid redundancy. It comprises a collection of hardware modules as basic building blocks, such as detectors, voters and preventers, that can then be automatically composed together and integrated into a target design to realize diverse hardware redundancy techniques selected by the user.

First, after the target design is converted to FIRRTL, the compiler framework reads the target design, generates an internal representation for that design, and reads the configuration file from the user similar to the flow in Section 4. Next, the internal representation is passed through compiler passes including this tool. The passes made for this tool automatically modify the internal representation to make it support the fault tolerant technique indicated in the configuration file. The processes of integrating each redundancy technique to a target design are described later in this section.

This work benefits from using FIRRTL as described in Section 4 and also a variety of compiler passes inside its hardware compiler framework such as optimization passes. It also gains from generating a fault tolerant design at RTL level, as the RTL file is still readable, and thus can be modified further manually, for example, to implement a specific detection based on a generic hardware redundancy technique [22], and the fault tolerant structures inserted into the design can be fed through RTL synthesis to satisfy timing constraints [27].

Double Modular Redundancy. The double modular redundancy technique is shown in Figure 3a. LABS supports double modular redundancy at register and module level. At the register level, all target registers are duplicated, and each pair of the target and duplicated registers is connected to a detector building block, which is used to compare their outputs. At the module level, all components inside the module are duplicated, and all wires connected to the output ports are compared with their duplicates using detectors. Each detector is generated dynamically with the appropriate width for each pair of outputs. If there are more than one detectors, the outputs of the detectors will be or-ed, and the or-ed output will be connected to the detect port, which is automatically added, and can be connected to the design to indicate whether there is an error detected or not, for example, to reset or trigger an interrupt signal [4]. The preventer can also be added when needed to, for example, hide the output of the original block when errors are being detected to avoid faulty outputs to be seen by attackers.

Triple Modular Redundancy. The triple modular redundancy technique is shown in Figure 3b. Our tool supports triple modular redundancy at register and module level. The automatic integration method is similar to that for double modular redundancy. It adds two duplicated blocks identical to the original block, generates and inserts majority voters with appropriate width, connects all blocks to the voter, and connects the majority voter to the output ports. Recently, there has been a study about tradeoffs between various majority voter designs [3]. The voter can be redesigned freely depending on the user's choice.

Temporal Redundancy. The temporal redundancy technique is shown in Figure 3c. Our tool supports temporal redundancy at the module level. There are three additional components: InputStorage, OutputStorage and Controller. The InputStorage stores inputs needed for recomputation. The OutputStorage stores the outputs from the design block, uses a detector to compare the outputs to detect errors, and connects it to the detect signal, which is automatically added. The preventer can also be used inside the OutputReg when it is needed. The user is required to indicate the name of the start signal, which is a signal to start a computation of the design block, and the ready signal from the design block, which is used to indicate that the computation is done. The controller uses these two signals to control the InputStorage and OutputStorage, and sends the modified ready signal as an output, that sends a signal to the output port when the computations are already computed twice.

These three redundancy techniques are orthogonal and can be combined to form hybrid redundancy techniques. For more information, the works [4] and [30] summarize countermeasures that can be deployed to protect against fault attacks.

Extensions for Data Integrity Verification. While automatic redundancy generation is built into our proposed tool, there is also the potential to add algorithm-dependent techniques, like checksum or parity functions, depending on the operation implemented. Techniques like these can allow for more efficient integrity checks as they do not require the duplication of work. Our modular methodology does not restrict the type of verification techniques that can be added to the automated workflow. The tool mainly aims to support redundancy techniques to protect against fault attacks, which the attacker tries to inject faults directly to the target design. There are also techniques to protect side channel attacks, which the attacker tries to observe electrical properties of the target circuit, for example SABL [35] and WDDL [36]

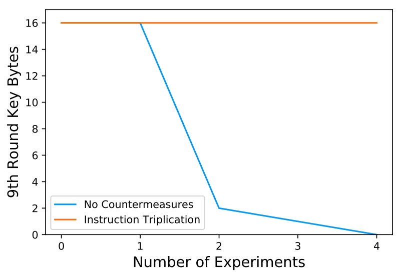


Figure 4: The result of the AES attack by Breier et al. [10] by injecting random faults into the instruction register on software AES with and without countermeasures.

6 EXPERIMENTAL SETUP

This tool was built with a number of tools and datasets. For the overall framework, we use FIRRTL version 1.2 and Chiffre for transforming the code, and Yosys 0.9 for Verilog to FIRRTL conversion. The AES software [43] was run on the Rocket core [2], and the AES hardware accelerator was implemented by [34]. The neural network was trained on the IRIS dataset [16] and implemented using Genann [42]. RSA algorithm implementation [31] was modified to implement the RSA-CRT algorithm. Verilog simulations are carried out with Synopsys VCS-MX K-2015.09-SP2-9, and the hardware was synthesized with Synopsys Design Compiler version P-2019.03-SP5 targeting a 22nm technology node. All experiments are run on two 14-core Intel Xeon Gold 6132 running at 2.6 GHz.

7 EXPERIMENTAL RESULTS

In the previous sections, we have outlined our automated methodology for a complete solution to better understand and mitigate hardware and software laser fault injection attacks. We began by outlining a number of key benchmarks typically targeted by laser-based attacks (Section 4), and followed up with automatic hardware generation to mitigate these attacks (Section 5). These previous sections demonstrated how, through automation, our methodology can help to close the loop from injection, mitigation and detection of laser faults.

In this section, we begin with a detailed example of a common use case, showing the steps needed from fault injection to mitigation techniques to demonstrate the effectiveness of the design created by the methodology. Next, we provide detailed output of the results when using our methodology on the key benchmarks included in this work. These results demonstrate how the detection and mitigation strategies provide robust results, and can be used for work in evaluating susceptibility to (and recovery from) laser fault injection attacks. Details into the run time (software simulation) overheads, and hardware overheads of using this methodology are shown.

7.1 Laser Fault Injection Attacks

Figure 4 shows an analysis of the outputs of the Rocket core running software AES being attacked by the AES attack by Breier et al. [10] with and without the instruction triplication countermeasure, a software countermeasure computing a critical instruction thrice [5]. The y-axis of the graph shows the number of bytes of the ninth round key left to be revealed. The target component is the instruction register at the execution stage of the pipeline, where random bit-flip faults are injected into when the xor instruction is being executed during the last *AddRoundKey* to skip the instruction. Four experiments are needed to reveal the entire last round secret key for the one without the countermeasure.

```
8000174c: xor a1,a1,a1           80001768: xori a1,a1,2
80001750: xor a4,a2,a3           8000176c: bne a4,a5,80001774
80001754: xor a5,a2,a3           80001770: xori a1,a1,4
80001758: xor a6,a2,a3           80001774: xori a1,a1,4
8000175c: bne a6,a4,80001764     80001778: bnez a1,80001780
80001760: xori a1,a1,1           8000177c: mv a6,a5
80001764: bne a6,a5,8000176c
```

Listing 2: The instruction triplication countermeasure implemented in the software implemented AES to protect against the AES attack by Breier et al. [10].

Listing 2 shows the assembly code when the instruction triplication countermeasure is specifically deployed to protect the xor instruction in RISC-V assembly similar to [5]. The register a6 will store the correct result from the exclusive-or between a byte of the ninth round temporary ciphertext and the last round key stored inside a2 and a3 respectively. Two extra registers a4 and a5 are used to compare their results with each other and a6 to correct errors. Using the same attack targeting the instruction at the address 0x80001750, it can be seen in Figure 4 that there are no bytes of the last round secret key revealed due to the countermeasure having its ability to mask faults similar to the hardware-based triple modular redundancy technique.

Figure 5 shows the AES attack by Giraud et al. [19] on the AES accelerator. User-defined faults are injected directly into the registers storing a ninth round temporary ciphertext at the beginning of the last round. The y-axis of the graph shows the number of bytes needed to be revealed. 47 experiments are needed to reveal the entire ninth round temporary ciphertext. This result also demonstrates that a design implemented in Verilog can be used with the framework.

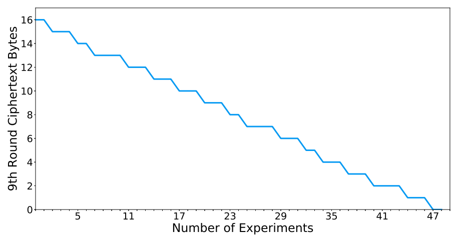


Figure 5: The result of the AES attack by Giraud et al. [19] by injecting user-defined faults into the AES accelerator. 47 experiments are needed to reveal the entire ninth round temporary ciphertext.

Figure 6 shows the comparison of the neural network running on the Rocket core, between ground-truth, inference without faults and inference with faults for each label. The neural network consists of three layers comprising four input neurons, four hidden neurons and three output neurons. The sigmoid function is used as an activation function for every neuron. The target location is the multiplexer inside the instruction cache that feeds instructions to the pipeline. The user-defined one-bit faults are injected to the

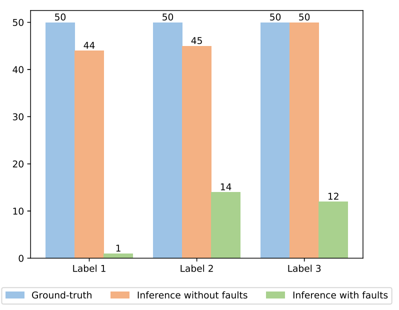


Figure 6: The number of the correct samples of the neural network running on the Rocket core during the deep learning attack by Breier et al. [10] compared with the ground-truth and inference without faults results.

(a) Elapsed time for behavioral simulation for all experiments.

Attacks	(m:ss)
AES [10]	8:30
AES [19]	:27
RSA-CRT [8]	2:26
NN [9]	6:09

(b) Elapsed time per step (AES Accel.)

Steps	(m:ss)
Fault-Tolerant Integration	:07
Hardware Fault Injection	:09
Simulation Compilation	:04
Behavioral Simulation	:01
Fault Analysis	:01
Synthesis	6:06

Table 3: Elapsed time for simulations.

location when the negation instruction of the sigmoid function of all the neurons is being sent to the pipeline to skip the instruction. The usual accuracy of the neural network is 92.67%, whereas, when attacked, the accuracy decreases to 18.0%.

Table 3a shows time duration needed for simulation for each benchmark to get outputs that lead to a successful attack. The time duration for the AES and deep learning attack by Breier et al., and AES attack by Giraud et al. is the simulation time needed to get the results shown in Figure 4, Figure 6 and Figure 5 respectively.

A previous work [40] proposes the state-of-the-art layout-based laser fault simulation that models faults injected by a laser at electrical level, which is at a lower level than logical level used in this work, and simulates non affected cells with gate level accuracy. Modelling laser faults at a lower level of abstraction is an alternative way to get a more realistic result, but takes a lot longer to simulate. From the data in [40], one laser shot can take one to six minutes to calculate induced faults. In Figure 6, 7 faults have to be injected into the Rocket core running the neural network program for each sample, meaning that 1050 laser shots are required to attack the core to test every sample. Moreover, gate level simulation of the core has to be completed to obtain the outputs for the exploitation phase, which can take 169 times longer than behavioral simulation [23]. The time duration needed to finish the entire simulation might not be desirable for our work.

7.2 Hardware-based redundancy defenses

An example of the outputs of the AES accelerator being attacked with different countermeasures deployed at module level is shown

```
Correct:  0x3ad77bb40d7a3660a89ecaf32466ef9
Faulty:   0x3ad77bb40d7a3697a89ecaf32466ef9
DMR:      0x00000000000000000000000000000000
Temporal: 0x00000000000000000000000000000000
TMR:      0x3ad77bb40d7a3660a89ecaf32466ef9
```

Figure 7: An example of the outputs of the AES accelerator being attacked by Giraud et al. [19] with different hardware countermeasures deployed at module level.

in Figure 7. It can be seen that the output without a countermeasure has one faulty byte at the 8th byte (shown in bold and underline text) due to a single bit fault injected directly into the register storing the ninth round temporary ciphertext in the beginning of the last round, which leads to secret key retrieval using the AES attack by Giraud et al. [19]. The outputs of the double modular redundancy and temporal redundancy are similar, which are all zeros due to the preventor used to hide a faulty ciphertext to be seen by the attackers. Note that the detect signals of these two countermeasures are also high, indicating that there is a difference between the outputs of the original design and its duplicate. The one with the triple modular redundancy shows the correct ciphertext due to its ability to rectify errors.

Table 4: Overheads of supported hardware-based redundancy techniques. Combs and Seqs stand for combinational and sequential standard cells respectively

Design	#Combs (Cells)	#Seqs (Cells)	Area (μm^2)	Power (mW)	Freq (GHz)
Original	17973	2472	10661	6.26	1.03
DMR	35973	4947	21526	12.63	1.03
Temporal	18843	2865	11478	7.70	1.03
TMR	53729	7416	32170	18.92	1.03
Hybrid	57228	8066	34353	20.54	1.03

Table 4 shows the overheads of each supported hardware-based redundancy technique. The target design for this experiment is the AES accelerator, and the target location is its entire core. The hybrid redundancy is done by applying the temporal and triple modular redundancy respectively.

7.3 End-to-End Laser Attack Evaluation

One of the main contributions of this methodology and benchmark collection is that it is now possible to perform a complete, full-circle evaluation of faults, coverage and mitigation strategies to allow one to quickly converge on a solution that provides the right trade-offs for the hardware design that needs to be protected. Our tool includes a set of commonly-used hardware blocks that can be protected. That said, we have made the work modular to provide an extensible solution for other error injection types, hardware designs and mitigation strategies.

In this example, we examine the mitigation of an attack on an AES accelerator [34]. More specifically, we would like to mitigate the issues caused by an AES attack which injects a one-bit fault in the last round of AES processing [19].

To start, we first indicate the register where we will be injecting a fault (`aes.aes_encipher_block.block_w3_reg`, Listing 1a), and the controller used to determine when the fault should occur

(Listing 1b). Finally, we configure the controller to watch for a trigger signal (`aes.aes_encipher_block.round_ctr_reg`, Listing 1c) and insert it into the module.

After running the fault injection framework, the resulting faulty bits can be seen in the output (Figure 7). After repeating this procedure for all of the necessary bits, the fault injection framework is able to recover the entire ninth round temporary ciphertext (Figure 5).

To prevent this from occurring in a new design, we select and integrate countermeasures needed to protect the circuit. In this example, we choose triple modular redundancy (TMR) to prevent the fault attacks. We update the LABS configuration (Listing 1d) to indicate which hardware block should be updated with redundancy, and re-run the hardware design flow (Figure 1). Our tool automatically introduces the redundant hardware, and generates a new design. After completing the workflow, we can see that the new design with TMR effectively mitigates the AES attack (Figure 7).

The original AES design, when synthesized, was ~10k μm^2, with a power consumption of 6.26 mW (Table 4). For the given frequency target, we can see the overheads for the TMR technique was approximately 3.02× in area and power. Given that the behavioral simulation for the Giraud technique is just 27 seconds (Table 3a), this evaluation technique is far faster than *Physical* or *Electrical* techniques, leading to significant savings with respect to evaluation of designs and understanding the trade-offs with respect to run time, power- and energy-efficiency. With the LABS framework, the time taken to evaluate whole-design fault mitigation techniques now becomes tractable.

Table 3b shows the time needed for each step in the flow to generate a test or fault-tolerant design for an experiment in Figure 5, compile and run behavioral simulation, run a fault analysis after getting outputs from experiments, which provides a graph in Figure 5, and synthesize the AES design with the triple modular redundancy countermeasure applying to the entire core.

8 RELATED WORK

In this section we describe several state-of-the-art tools that enable circuit designers to simulate laser fault injection and automate defensive integration during design stages.

Laser-Induced Fault Simulation Methodology. The authors in [40] propose a layout-based laser fault simulation modelled at the electrical level. This work uses standard commercial CAD tools, and takes into account IR drop effects. In [25] the authors propose a layout-based fault simulation using an HDL/SPICE co-simulator. The cells under virtual laser illumination are simulated using detailed full-transistor level netlists in SPICE. Voltage spikes are modelled to mimic laser effects. It simulates the rest of the target design using Verilog netlists. To improve simulation performance, the work in [26] introduces a multi-level simulator that simulates at the electrical level only the area that is affected by the laser, while the rest of the target system is simulated at the gate level. Modelling laser induced faults at the electrical level provides for realistic results that are based on current source, however electrical-level simulations are time-consuming and require proprietary tools to implement. With LABS we employ open-source tools to model faults generated by a laser at the logical level that is orders of magnitude faster.

Laser Fault Model. The work in [29] introduces the first RTL laser fault model that reduces fault space compared to random multi-bit fault injection. It relies on an assumption that functional relation between elements inside the design is not changed through the digital design flow. The goal of this work was to reduce the amount of work needed to be done during random multi-bit fault injection while our work aims to generate specific attacks to compromise features of various safety-critical applications.

Physical Attack Simulation. In [14] the authors present a commercial tool called VIRTUALYZR®. The tool supports both fault injection attacks and side channel attacks simulation in various abstraction levels. For the fault injections, it uses logical level fault models such as stuck-at, bit-flip and random faults to inject faults into a location that can be at bit level, variable level or random. Our work bears similarities to the features claimed by [14], however we offer an open-source framework based on the open-source hardware construction language, FIRRTL, and its hardware compiler framework. More importantly, our work goes beyond attacks and introduces a tool that automatically integrates fault tolerant structures into a target design.

Automatic Insertion of Fault Tolerance Structures. The work in [27] presents a tool that automatically integrates concurrent error detection in Verilog RTL, supporting four options. They can then be applied to finite state machines indicated by the user, and a coverage evaluation tool to evaluate the coverage of the options. The authors in [7] propose an industrial framework that generates single-event upsets, and integrates fault tolerant structures into a VHDL RTL design. In [6], the authors propose an approach for a commercial ASIC design flow that integrates triple modular redundancy structures into a target design at netlist level. Our work is implemented as compiler passes based on an open-source hardware compiler framework comprising a variety of transformation passes such as optimization passes. Therefore, our passes can be used seamlessly with other related work such as Chiffre. It also transforms a target design in the hardware intermediate representation, FIRRTL, instead of one of the hardware description languages directly, which makes LABS not limited to only one specific hardware design language or a specific commercial tool.

9 CONCLUSION

This work proposes a framework to simulate laser fault injection attacks from various applications. This work also provides the hardware-based redundancy integration tool integrated into the FIRRTL compiler that adds hardware-based redundancy techniques into a target design without manual modifications. The framework and the tool together will automate the entire security evaluation loop, both attacks and defenses, that will help facilitate pre-silicon security evaluation against laser fault injection. The laser attack benchmark suite can be found on the website [1].

ACKNOWLEDGMENTS

The authors acknowledge the support from the Singapore National Research Foundation (“SOCure” grant NRF2018NCR-NCR002-0001 – www.green-ic.org/socure).

REFERENCES

[1] Burin Amornpaisannon, Andreas Diavastos, Li-Shiuan Peh, and Trevor E. Carlson. Laser Attack Benchmark Suite. https://github.com/nus-labs/labs.

[2] Krste Asanović, Rimas Avizienis, Jonathan Bachrach, Scott Beamer, David Biancolin, Christopher Celio, Henry Cook, Daniel Dabbelt, John Hauser, Adam Izraelevitz, Sagar Karandikar, Ben Keller, Donggyu Kim, John Koenig, Yunsup Lee, Eric Love, Martin Maas, Albert Magyar, Howard Mao, Miquel Moreto, Albert Ou, David A. Patterson, Brian Richards, Colin Schmidt, Stephen Twigg, Huy Vo, and Andrew Waterman. 2016. *The Rocket Chip Generator.* Technical Report UCB/EECS-2016-17. EECS Department, University of California, Berkeley. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html

[3] Padmanabhan Balasubramanian and Nikos E. Mastorakis. 2016. Power, Delay and Area Comparisons of Majority Voters relevant to TMR Architectures. *CoRR* abs/1603.07964 (2016). arXiv:1603.07964 http://arxiv.org/abs/1603.07964

[4] Hagai Bar-El, Hamid Choukri, David Naccache, Michael Tunstall, and Claire Whelan. 2006. The Sorcerer's Apprentice Guide to Fault Attacks. In *Proceedings of the IEEE*, Vol. 94. 370–382. https://doi.org/10.1109/JPROC.2005.862424

[5] Alessandro Barenghi, Luca Breveglieri, Israel Koren, Gerardo Pelosi, and Francesco Regazzoni. 2010. Countermeasures against Fault Attacks on Software Implemented AES: Effectiveness and Cost. In *WESS*. https://doi.org/10.1145/1873548.1873555

[6] Luis Alberto Contreras Benites and Fernanda Lima Kastensmidt. 2018. Automated design flow for applying Triple Modular Redundancy (TMR) in complex digital circuits. In *LATS*. 1–4. https://doi.org/10.1109/LATW.2018.8349668

[7] Luis Berrojo, Fulvio Corno, Luis Entrena, Isabel Gonzalez, Celia López, Matteo Sonza Reorda, and Giovanni Squillero. 2002. An industrial environment for high-level fault-tolerant structures insertion and validation. In *VTS*. 229–236. https://doi.org/10.1109/VTS.2002.1011143

[8] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. 2001. On the Importance of Eliminating Errors in Cryptographic Computations. In *J. Cryptology*, Vol. 14. 101–119. https://doi.org/10.1007/s001450010016

[9] Jakub Breier, Xiaolu Hou, Dirmanto Jap, Lei Ma, Shivam Bhasin, and Yang Liu. 2018. Practical Fault Attack on Deep Neural Networks. In *CCS*. 2204–2206. https://doi.org/10.1145/3243734.3278519

[10] Jakub Breier, Dirmanto Jap, and Chien-Ning Chen. 2015. Laser Profiling for the Back-Side Fault Attacks: With a Practical Laser Skip Instruction Attack on AES. In *CPSS*. 99–103. https://doi.org/10.1145/2732198.2732206

[11] Jakub Breier, Dirmanto Jap, and Chien-Ning Chen. 2018. *Laser-Based Fault Injection on Microcontrollers*. Springer Singapore, 81–110. https://doi.org/10.1007/978-981-10-1387-4_5

[12] Cadence. Spectre eXtensive Partitioning Simulator. https://www.cadence.com/en_US/home/tools/custom-ic-analog-rf-design/circuit-simulation/spectre-extensive-partitioning-simulator-xps.html.

[13] Cadence. Voltus IC Power Integrity Solution. https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/silicon-signoff/voltus-ic-power-integrity-solution.html.

[14] Kais Chibani, Adrien Facon, Sylvain Guilley, Damien Marion, Yves Mathieu, Laurent Sauvage, Youssef Souissi, and Sofiane Takarabt. 2019. *Fault Analysis Assisted by Simulation.* Springer International Publishing, 263–277. https://doi.org/10.1007/978-3-030-11333-9_12

[15] Moslem Didehban, Aviral Shrivastava, and Sai Ram Dheeraj Lokam. 2017. NEMESIS: A software approach for computing in presence of soft errors. In *ICCAD*. 297–304. https://doi.org/10.1109/ICCAD.2017.8203792

[16] Dheeru Dua and Casey Graff. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml

[17] Jean-Max Dutertre, Vincent Beroulle, Philippe Candelier, Stephan De Castro, Louis-Barthelemy Faber, Marie-Lise Flottes, Philippe Gendrier, David Hély, Regis Leveugle, Paolo Maistri, Giorgio Di Natale, Athanasios Papadimitriou, and Bruno Rouzeyre. 2018. Laser Fault Injection at the CMOS 28 nm Technology Node: an Analysis of the Fault Model. In *FDTC*. 1–6. https://doi.org/10.1109/FDTC.2018.00009

[18] Schuyler Eldridge, Alper Buyuktosunoglu, and Pradip Bose. 2018. Chiffre: A Configurable Hardware Fault Injection Framework for RISC-V Systems. In *CARRV*.

[19] Christophe Giraud. 2005. DFA on AES. In *Advanced Encryption Standard – AES*, Hans Dobbertin, Vincent Rijmen, and Aleksandra Sowa (Eds.). Springer Berlin Heidelberg, 27–41. https://doi.org/10.1007/11506447_4

[20] Adam Izraelevitz, Jack Koenig, Patrick Li, Richard Lin, Angie Wang, Albert Magyar, Donggyu Kim, Colin Schmidt, Chick Markley, Jim Lawson, and Jonathan Bachrach. 2017. Reusability is FIRRTL ground: Hardware construction languages, compiler frameworks, and transformations. In *ICCAD*. 209–216. https://doi.org/10.1109/ICCAD.2017.8203780

[21] Allan H. Johnston. 1993. Charge generation and collection in p-n junctions excited with pulsed infrared lasers. In *TNS*. 1694–1702. https://doi.org/10.1109/23.273491

[22] Marc Joye, Pascal Manet, and Jean-Baptiste Rigaud. 2007. Strengthening hardware AES implementations against fault attacks. In *IET Information Security*. 106–110. https://doi.org/10.1049/iet-ifs:20060163

[23] Dusung Kim, Maciej Ciesielski, Kyuho Shim, and Seiyang Yang. 2011. Temporal parallel simulation: A fast gate-level HDL simulation using higher level models. In *DATE*. 1–6. https://doi.org/10.1109/DATE.2011.5763251

[24] Francois Koeune and François-Xavier Standaert. 2004. A Tutorial on Physical Security and Side-Channel Attacks. In *LNCS*. 78–108. https://doi.org/10.1007/11554578_3

[25] Huiyun Li and Simon Moore. 2006. Security evaluation at design time against optical fault injection attacks. In *IEE Proceedings - Information Security*. 3–11. https://doi.org/10.1049/ip-ifs:20055021

[26] Feng Lu, Giorgio Di Natale, Marie-Lise Flottes, and Bruno Rouzeyre. 2013. Laser-Induced Fault Simulation. In *DSD*. 609–614. https://doi.org/10.1109/DSD.2013.72

[27] Kartik Mohanram, Ch V. Phani Krishna, and Nur A. Touba. 2002. A methodology for automated insertion of concurrent error detection hardware in synthesizable Verilog RTL. In *ISCAS*. https://doi.org/10.1109/ISCAS.2002.1009906

[28] National Institute of Standards and Technology. Advanced Encryption Standard (AES), NIST FIPS PUB 197.

[29] Athanasios Papadimitriou, David Hély andVincent Beroulle, Paolo Maistri, and Régis Leveugle. 2014. A multiple fault injection methodology based on cone partitioning towards RTL modeling of laser attacks. In *DATE*. 1–4. https://doi.org/10.7873/DATE.2014.219

[30] Sikhar Patranabis and Debdeep Mukhopadhyay. 2018. *Classical Countermeasures Against Differential Fault Analysis.* Springer Singapore, 171–182. https://doi.org/10.1007/978-981-10-1387-4_8

[31] Amruth Pillai. RSA Algorithm in C. https://gist.github.com/AmruthPillai/42f4fef15bd2591aeddccae03b31ab25.

[32] Dhiman Saha, Debdeep Mukhopadhyay, and Dipanwita Roy Chowdhury. 2009. A Diagonal Fault Attack on the Advanced Encryption Standard.. In *IACR Cryptology ePrint Archive.*

[33] Bodo Selmke, Johann Heyszl, and Georg Sigl. 2016. Attack on a DFA Protected AES by Simultaneous Laser Fault Injections. In *FDTC*. 36–46. https://doi.org/10.1109/FDTC.2016.16

[34] Joachim Strömbergson. Verilog implementation of the symmetric block cipher AES. https://github.com/secworks/aes.

[35] Kris Tiri, Moonmoon Akmal, and Ingrid Verbauwhede. 2002. A dynamic and differential CMOS logic with signal independent power consumption to withstand differential power analysis on smart cards. In *ESSCIRC*. 403–406.

[36] Kris Tiri and Ingrid Verbauwhede. 2004. A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation. In *DATE*. 246–251.

[37] Elena Trichina and Roman Korkikyan. 2010. Multi Fault Laser Attacks on Protected CRT-RSA. In *FDTC*. 75–86. https://doi.org/10.1109/FDTC.2010.14

[38] Jasper G. J. van Woudenberg, Marc F. Witteman, and Federico Menarini. 2011. Practical Optical Fault Injection on Secure Microcontrollers. In *FDTC*. 91–99. https://doi.org/10.1109/FDTC.2011.12

[39] Pierre Vanhauwaert, Paolo Maistri, Regis Leveugle, Athanasios Papadimitriou, David Hely, and Vincent Beroulle. 2014. On error models for RTL security evaluations. In *DTIS*. 1–6. https://doi.org/10.1109/DTIS.2014.6850666

[40] Raphael A.C. Viera, Jean-Max Dutertre, Philippe Maurine, and Rodrigo Possamai Bastos. 2018. Standard CAD Tool-Based Method for Simulation of Laser-Induced Faults in Large-Scale Circuits. In *ISPD*. 160–167. https://doi.org/10.1145/3177540.3178243

[41] Raphael Andreoni Camponogara Viera, Jean-Max Dutertre, Rodrigo Possamai Bastos, and Philippe Maurine. 2017. Role of Laser-Induced IR Drops in the Occurrence of Faults: Assessment and Simulation. In *DSD*. 252–259. https://doi.org/10.1109/DSD.2017.43

[42] Lewis Van Winkle. C Neural Network Library: Genann. https://github.com/codeplea/genann.

[43] Cecelia Wisniewska. C++ implementation of a 128-bit AES encryption/decryption tool. https://github.com/ceceww.

[44] Clifford Wolf. Yosys Open SYnthesis Suite. http://www.clifford.at/yosys/.