Securing RSA Hardware Accelerators through Residue Checking

Ana Lasheras[a], Ramon Canal[a], Eva Rodríguez[a], Luca Cassano[b,*]

[a]*Departament d'Arquitectura de Computadors – Universitat Politecnica de Catalunya, Barcelona, Spain*
[b]*Dipartimento di Elettronica, Informazione e Bioingegneria – Politecnico di Milano, Italy*

Abstract

Circuits for the hardware acceleration of cryptographic algorithms are ubiquitously deployed in consumer and industrial products. Although being secure from a mathematical point of view, such accelerators may expose several vulnerabilities strictly related to the hardware implementation. Differential fault analysis (DFA) and hardware Trojan horses (HWTs) may be exploited to steal secret information from the circuit or to interfere with its nominal functioning. It is therefore important to protect cryptographic hardware accelerators against such attacks in a efficient way. In this paper, we propose a lightweight technique for protecting circuits implementing the RSA algorithm against DFA and HWTs at runtime. The proposed solution relies on residue checking which is a well-known technique belonging to traditional fault tolerance. Residue checking is here applied to RSA circuits in order to detect any modification of the output of the circuit possibly induced by the occurrence of a fault or the activation of a HWT. When this happens the protection technique reacts to the attack by obfuscating the circuit's output (i.e. generating a random output). An experimental campaign (99% confidence and 1% error) demonstrated that, when dealing with DFA, the proposed solution detected 100% of the fault attacks that leaked information to the attacker. Moreover, we applied the proposed technique to all the HWT infected implementations of the RSA algorithm available in the Trust-Hub benchmark suite achieving a 100% HWT detection. The overhead introduced by the proposed solution is a maximum area increase below 3%, about 18% dynamic power consumption increase while it has no impact on the operating frequency.

Keywords: Cryptographic Accelerators, Differential Fault Analysis, Fault Attacks, Hardware Trojans, Hardware Security, Residue Checking, RSA, Third Party Intellectual Property Cores (3PIPs)

1. Introduction

Hardware accelerators for cryptographic algorithms are widely employed in those consumer products, such as smart phones and smart cards, that expose both security and performance requirements. Although modern cryptographic algorithms are secure from the mathematical point of view [1], their implementations often suffer from a number of vulnerabilities. In particular, *differential fault analysis (DFA)* has been demonstrated to be a very efficient way to leak secret information from cryptographic hardware accelerators [2, 3].

Moreover, the increasing complexity of integrated circuits (ICs) and the seek for low production cost and short time-to-market, led to the globalization of the design and fabrication process of silicon devices [4] also in the market of cryptographic hardware accelerators. More and more often the design of some of the hardware modules is outsourced, third-party intellectual property cores (3PIPs) are bought, and sometimes also masks and the final chip fabrication are outsourced [5]. On the one hand, this globalization allows for a significant reduction of design cost and time, but it comes with a significant loss of trust in the delivered ICs [6].

It is very hard to ensure the trustworthiness of all the parties involved in such a globalized supply chain. As a consequence, the product is exposed to a huge number of threats, among which overproduction [7], counterfeiting [8], 3PIPs licenses violation and abuse [9] and Hardware Trojan Horses (HWTs) [10]. A HWT is a very-hard-to-detect modification of a design that stays silent most of the time, while in specific (usually rare) conditions it alters the behavior of the system or it steals secret information. HWTs may be inserted by vendors in the purchased 3PIP core [11], by employees in the developed HDL code, by CAD tools [12] and by mask

*Corresponding author
Email addresses: ana.lasheras@est.fib.upc.edu (Ana Lasheras), rcanal@ac.upc.edu (Ramon Canal), rcanal@ac.upc.edu (Eva Rodríguez), luca.cassano@polimi.it (Luca Cassano)

providers and silicon foundries in the final layout [13].

This paper focuses on *RSA (Rivest-Shamir-Adleman)* which is a widely adopted public-key encryption algorithm [14]. We propose a runtime solution aimed at protecting RSA hardware accelerators against DFA and HWTs. By monitoring circuit's behavior, the proposed solution is able to detect at runtime a tentative fault attack and the activation of a HWT. Each time a possible threat condition is detected, the proposed solution substitutes the output of the circuit with a random one in order not to expose sensitive information that could then be used by the attacker to retrieve the secret key.

The proposed attack detection technique borrows residue checking from classical fault detection techniques. The basic idea behind our proposal is to pair the *conventional* RSA implementation with a replica of the RSA algorithm (dubbed the *modulo RSA*) that uses a smaller number of input bits. The inputs of the modulo RSA are the results of a modulo operation of the inputs of the conventional RSA (i.e. their residue values). The properties of the modulo operation ensure that, at the end of the execution of both replicas, the result of the modulo operation on the output of the conventional RSA will match the output of the modulo RSA only if no alteration of the nominal processing occurred, e.g., no HWT activated and changed the output of the nominal RSA or no faults occurred during the nominal RSA processing. Whenever a perturbation of the behavior of the nominal RSA happens and modifies its output, this check will fail. This would be the case of a HWT activation or the occurrence of an either random or intentional fault. It is worth mentioning that the proposed technique works at the logic-level, thus it can be applied regardless of the underlying technology, e.g., FPGA or ASIC. A preliminary version of the proposed approach has been applied to protect AES, SHA-2 and RSA from fault attacks in [15, 16].

An experimental campaign (99% confidence and 1% error) demonstrated that the proposed countermeasure allowed to detect 100% of the fault attacks that leaked information to the attacker plus, all the HWTs in all the RSA infested circuits available in the TrustHub benchmark [17]. The cost of our solution is an area overhead below 3%, about 18% power consumption increase and no operating frequency reduction.

The remainder of this paper is organized as follows: Section 2 introduces the basics of the RSA algorithm; Section 3 discusses the considered threat models; Section 4 presents the proposed protection technique; Section 5 reports the results from an experimental campaign and a qualitative comparison against state of the art approaches, Section 6 presents a security analysis and discusses the limitations of the proposed solution; Section 7 presents the related work. Finally, Section 8 concludes the paper.

2. The RSA algorithm

RSA (Rivest-Shamir-Adleman) is a public-key cryptographic (or asymmetric) algorithm [14]. Two keys are employed: the *public* key is used for encryption and the *private* key, which is kept secret by the owner, is used for decryption. In the remainder of the paper, we will use the following notation: m denotes the plain message (also dubbed as *plaintext*), c denotes the encrypted message (also dubbed as *ciphertext*), s denotes a signature when RSA is used for integrity instead of confidentiality, $n = p * q$ denotes the public modulus, where p and q are large prime numbers, e denotes the public exponent, and d the private (secret) exponent. The public key consists of n and e, while the private key of the secret d. It is worth mentioning that e, d and n must be chosen such that for all integers m (with $0 \leq m \leq n$) the following equation holds:

$$(m^e)^d \equiv m \bmod n \tag{1}$$

as well as:

$$(m^d)^e \equiv m \bmod n \tag{2}$$

For the sake of space, we do not mention how e, d and n are generated. Once e and n have been distributed, they can be used for encryption. Given a plaintext m, the corresponding cyphertext c can be computed as:

$$c \equiv m^e \bmod n \tag{3}$$

The only way to decrypt c is by using the private key associated with the public key used to encrypt the original plaintext. In other words, only the owner of d can decrypt c by computing:

$$c^d \equiv (m^e)^d \equiv m \bmod n \tag{4}$$

In case RSA is used for generating a message signature s the following equation is computed:

$$s \equiv m^d \bmod n \tag{5}$$

The signature s associated with a message m is considered to be valid if and only if:

$$s^e \equiv (m^d)^e \equiv m \bmod n \tag{6}$$

This means that the signature is valid if and only if it has been produced with the right private key (and thus by the right person) and the message has not been modified.

3. Threat model

Here we present the considered threat models in terms of fault attacks specific for RSA accelerators and possible HWTs.

3.1. Fault Attacks against RSA

As reported in [18], fault injection techniques can be divided into *low-cost* (below 3,000US$) and *high-cost* (3,000US$ up to millions of dollars). High cost techniques, that rely on very precise laser or ion beams or light pulse generators, allow the attacker to very accurately determine the injection location and time. It is extremely difficult to protect the system in case the attacker has access to such fault injection facilities.

On the other hand, low cost fault injection may be carried out by down- or over-scaling power supply, tampering the clock signal and over-heating the circuit. Such techniques only provide either time or space accuracy, but, given their low cost, and their wide applicability, are the ones addressed by most proposed circuit protection techniques. When dealing with RSA two families of low cost attacks exist: one aims at either factoring n (to discover the prime numbers p and q) or directly recovering d; the other aims at decrypting an encrypted message c without knowing d. A detailed discussion on DFA against RSA (and other cryptographic algorithms) may be found in [18].

The first family of attacks aims at retrieving the private key (or the prime numbers from which it is generated) during a decryption or the computation of a signature. The first type of attacks belonging to this family attempts to discover the entire secret information with a single injected fault. Such attack, dubbed the *Bellcore attack* [19], injects the fault trying to induce the system to compute $\widetilde{s}$ instead of s, being s the expected fault-free signature and $\widetilde{s} = s + \delta$. After $\widetilde{s}$ and s calculation, the attacker is able to retrieve one of the secret prime numbers from which the key has been generated (and thus also the secret key) by simply calculating the greatest common divisor between $(\widetilde{s} - s)$ and n. A similar attack has been proposed in [20] where fault injection happens during signature checking: this attack allows to recover the prime number by calculating the greatest common divisor between $(\widetilde{s^e} - m)$ and n. The great advantage of such attack w.r.t. the Bellcore attack is not to require the fault-free signature s.

A second type of attacks belonging to the first family aims at leaking one bit of the secret key at a time. Such attacks work under the following assumptions: i) the attacker has arbitrary access to the circuit; ii) the attacker can select the circuit's input; iii) there is no limitation to the number of fault injection attempts the attacker can perform (in other words, the fault injection activity is assumed not to be destructive). An attack belonging to this family has been proposed in [21]. Several faults are injected during several signature computations where each fault allows to leak a single bit of the private key. Indeed, a fault injected during signature computation may be such that the corrupted output signature is either $\widetilde{s} = s^{d-2^i}$ or $\widetilde{s} = s^{d+2^i}$ (depending on whether the induced bit-flip was a 0-to-1 or a 1-to-0) where $i \in [0, v-1]$ and v is the number of bits of the private key. Once $\widetilde{s}$ has been computed, the attacker can calculate the i^{th} bit of the secret key as either $m^{2^i} \bmod n = s/\widetilde{s} \bmod n$ or $m^{2^i} \bmod n = \widetilde{s}/s \bmod n$.

In the last type of attacks belonging to this first family (the so-called *safe error attacks* [22, 23]) the attacker only exploits the knowledge of whether the injected fault had an effect on the produced output or not to retrieve the key. More in details, in case a fault is injected in a register storing one of the bits of the key, the produced output will be unaltered w.r.t. the expected output in case the bit of the key was 0 (flipped to 1 by the fault). Conversely, the produced output will differ from the expected output in case the bit of the key was 1 (flipped to 0 by the fault). This attack is extremely effective but it has been demonstrated to be impractical [24]. Indeed, the fault injector is required to have both location and timing precision at the same time in order the safe error attack to be successfully carried out.

The goal of the second family of attacks to RSA is decrypting an encrypted message without the private key. This is achieved by injecting faults during encryption [24]. Like in the Bellcore attack, the retrieved corrupted cyphertext is either $\widetilde{c} = c^{e-2^i}$ or $\widetilde{c} = c^{e+2^i}$ where, again, c is the expected cyphertext, $i \in [0, v-1]$ and v is the number of bits of the public key. Given the correct cyphertext c and the faulty one $\widetilde{c}$, and after computing inverses over $\mathbb{Z}$, the attacker can then calculate $c \cdot \widetilde{c}^{-1} \bmod n$, thus, easily retrieving the plaintext associated with c.

The protection solution proposed in the current paper represents an effective and lightweight countermeasure against all the previously mentioned types of fault attacks except for the safe error one that, as we mentioned, is very unlikely to be deployed in a real-world scenario.

3.2. Hardware Trojans

A HWT is a very-hard-to-detect malicious modification of a digital circuit/system. HWTs may be classified according to the untrusted entity in the circuit's supply chain that inserts the Trojan, i.e., IP vendors, internal designers, CAD tools, silicon foundry.

Based on the triggering mechanism, HWTs may be divided in *triggered* HWTs and *always-on* ones. As the name suggests, always-on HWTs are always active in the infested circuit. Triggered HWTs, on the other hand, become active as soon as their triggering condition is verified. The activation of a triggered HWT may be controlled by a specific (generally rare) configuration of the signals of the circuit, by internal physical conditions of the circuit (e.g., temperature and voltage) or by externally received signals and messages.

A final classification may be done based on the effect of the HWT after its activation [25]. In this sense, HWTs may be divided into:

- *change-functionality*: HWTs that modify the behavior of the system.
- *denial-of-service*: HWTs that halt the functioning of the system by draining power supply, deactivating I/O interfaces or overloading processing units.
- *information-stealing*: HWTs that leak secret information and send it to the output or through a covert side-channel.

In this work, the entities considered as untrusted are the external providers of 3PIPs implementing the RSA HW accelerator. On the other hand, the system integrator (and thus the modulo RSA designers) and silicon foundries are assumed to be trusted. The detection technique presented in this paper addresses all those HWTs in externally purchased IP cores that modify the internal signals of the circuit to change or deny its functionality or to leak information, irrespective of the triggering mechanism (both triggered and always-on HWTs are considered). HWTs that modify the internal signals of the circuit without modifying its output signals are not considered since they can neither modify/halt the normal functioning of the circuit nor use the available output signals to sneak stolen information. Similarly, HWTs that leak information by sending it to a covert side-channel and HWTs that deny the functioning of the circuit by acting on the power supply or the communication interfaces are beyond the scope of this work.

Collusion between 3PIPs belonging to the same vendor, i.e., a core activates a HWT installed in another core, is also taken into account by the proposed technique as long as the unwanted malicious interaction between the two untrusted 3PIPs falls in one of the above mentioned modifications of the circuit's behavior.

As a last consideration, it has to be mentioned that the applicability of the proposed detection technique is constrained by the availability of either the HDL source code or the netlist of the considered 3PIP. In other words, the proposed technique is applicable to white-box netlists. We believe that this is a reasonable assumption in a scenario where the IP provider may disclose IP cores to allow the customer to verify the fulfilment of a previously agreed trustworthiness level. Such assumption is the same as in several hardware security-related papers working both at the HDL [26, 27] and netlist level [28, 29]. Finally, in the increasing trend of Open Hardware, it is common to share IP designs. As of real-world commercial products, the Cobham Gaisler LEON [30] and NOEL [31] CPUs, are also sold as "white-box" IP cores.

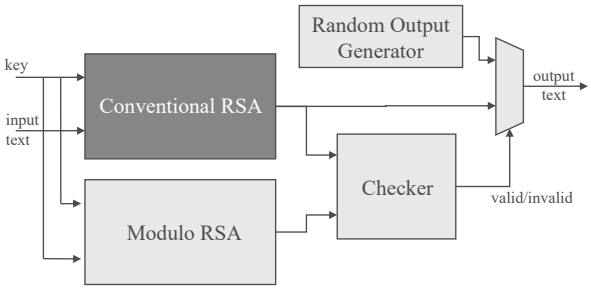

Figure 1: Block Diagram of RSA + Protection Circuit

4. The proposed Protection Architecture

We propose to enhance a system hosting a 3PIP that implements the RSA algorithm (light grey block in Figure 1) with a surrounding architecture (dark grey blocks in Figure 1). Our proposal is aimed at detecting at runtime the occurrence of a fault attack or the activation a HWT. More in details, we pair the purchased 3PIP implementation of the RSA algorithm (the *conventional RSA*) with an ad hoc designed much smaller replica (the *modulo RSA*). The outputs of the two replicas are then checked: in case no difference is observed, no fault attack or Trojan activation is assumed to have occurred and thus the output of the conventional RSA is forwarded as the output of the system. In case the two output values are different, a random output is generated and forwarded in order to make differential fault analysis (and more in general, information stealing) unfeasible.

It is worth mentioning that we here consider the conventional RSA as the component possibly under attack. Indeed, when considering fault attacks, the conventional RSA is the only component that executes the actual algorithm, and thus it is the only source of information for the attacker. On the other hand, no information would

be retrieved by attacking the modulo RSA. Similarly, when considering HWTs, the conventional RSA is the only externally purchased 3PIP and thus it is the only untrusted component in the system. On the other hand, the modulo RSA is assumed to be trusted since it is implemented by the internal design team (as previously discussed, we here consider the design team, the employed CAD tools and the foundry to be trusted). In case the attacker injects faults in the modulo RSA, instead of the conventional RSA, the faults would also be detected thus leading the system to output a random value instead of the uncorrupted output of the conventional RSA. This may allow the attacker to perform a denial-of-service attack to the system. Since this attack would not expose any secret information to the attacker, we leave this situation out of the scope of the paper. Finally, a severe source of threat would be the injection of multiple faults, but this would be discussed in depth later, in the Section 6.

In the remainder of this section, we provide additional details about the proposed protection technique. We first recall the basics of residue checking and how to use it to detect signals modifications and we then discuss how to select the best modulo value and how a HW implementation of RSA can be enhanced with the proposed residue checking-based detection mechanism.

4.1. Detecting Signal Modifications through Residue Checking

Given two positive integers a and n (where n is dubbed the *modulo* value), we can calculate a as:

$$a = n * q + r \tag{7}$$

where both q and r are positive integers (q is the *quotient* and r is the *residue*). The operation `modulo(a, n)` is defined as the residue of the integer division of a by n (represented as $|a|_n$) and it returns r.

Residue Checking is a lightweight mathematical mechanism for statically validating the results produced by arithmetic units. It detects changes on the output data based on the calculation of the residue. For instance, an addition can be checked by testing the equality of Equation 8.

$$|a + b|_n = ||a|_n + |b|_n|_n \tag{8}$$

Indeed, if both sides of the equation are equal, no error occurred. On the other hand, the occurrence of a fault during the processing would for sure make the check fail. Thus, when used to detect errors, residue checking guarantees that there are no false positives.

Residue checking works as described for any arithmetical operator. As a consequence, it also works for the modular exponentiation that is a key component of the RSA algorithm. On the other hand, it is worth noting that residue checking does not work for logical operators [32].

4.2. Selection of the modulo value

One of the key aspects of a fault detection mechanism based on residue checking is the selection of the modulo value. The chosen modulo value strongly affects both the fault detection capability and the introduced overhead. Breveglieri et al. [33] showed that the detection capability depends on the magnitude of the modulo value when transmitting information (i.e. no arithmetic operation involved between residue values). Below, we describe the particularities of power of 2 and non power of 2 values considering the implementation overhead and the detection capabilities. Figure 4 in Section 3 shows the fault detection percentage of for all the modulo values analyzed.

A modulo value n that is a power of 2 (i.e. $n = 2^w$) can be implemented as a logical masking operation, thus reducing the introduced overhead. When the modulo value is a power of 2, given an integer a the modulo operation can be expressed as:

$$|a|_n = a \,\&\, (2^w - 1) \tag{9}$$

Consequently, the residue value can be expressed in $w-1$ bits (if treated as an unsigned integer when designing the circuit). This choice significantly simplifies the implementation while, on the other hand, it makes the fault detection much less effective. For any power of 2 modulo value, the residue checking operation will miss any modification of the upper bits (i.e. the masked bits). Therefore, the smaller the modulo value, the smaller the introduced overhead but also the lower the fault detection capability (as we will see in in Section 3).

When the modulo is a non-power of 2 value, the calculation of the residue requires the use of all the bits in a and the fault detection capability does not depend on the chosen modulo value. A naive implementation of the modulo operation for non power of 2 modulo values simply computes the division and then makes a subtraction. Such implementation, although being straightforward, introduces a very high area overhead. Much more optimized implementations with fewer hardware requirements exist [34]. In this line, we rely on the optimized implementation of the modulo operation provided by the Xilinx Vivado™ [35] synthetizer.

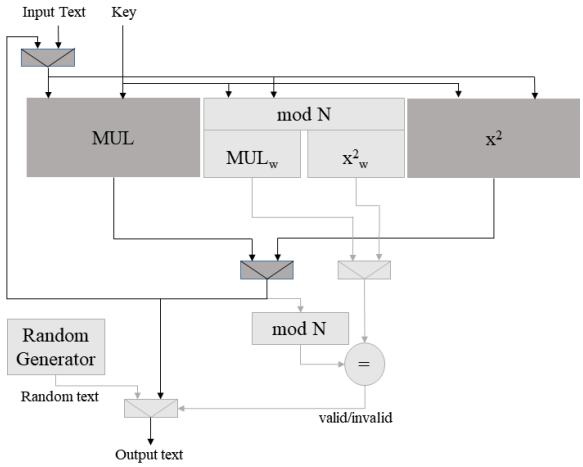

Figure 2: Block Diagram of RSA + Protection Circuit

4.3. Architectural details

The implementation of the conventional RSA is based on the implementation of the modular multiplication operation. As it has been discussed in Section 2, the RSA encryption performs the exponent operation shown in equation (3) and the decryption performs the exponent operation shown in equation (4). Given two positive integers x and y, x^y (referenced in the text as `Pow(x, y)`) can be computed as the result of the following recursive algorithm:

$$\text{Pow}(x, y) = \begin{cases} x, & \text{if } y = 1 \\ \text{Pow}(x^2, y/2), & \text{if } y \text{ is even} \\ x \times \text{Pow}(x^2, (y-1)/2), & \text{if } y > 2 \text{ is odd} \end{cases}$$

This algorithm is much faster than the ordinary method to perform the exponent calculation. Indeed, when multiplying x by itself, calculating x^y requires y operations. On the other hand, when implementing the algorithm shown above, only $log_2(y)$ operations are needed. The circuit implementing the conventional RSA based on such an algorithm for the exponent calculation only counts one multiplier and one square. The feedback loop that brings the output back to the input multiplexer (also shown in Figure 2) implements the recursivity of the algorithm.

A high-level representation of the overall secured system is shown in Figure 2: dark grey blocks belong to the conventional RSA while the light grey are added to implement the modulo RSA. More in details, the modulo RSA is composed of: i) the modulo operation (*modN*), ii) the reduced size multiplicator (MUL_w), iii) the square (X^2_w) block, iv) the necessary muxes, v) the comparator, and vi) the random output generator . It is worth mentioning that $w = \lceil log_2(n) \rceil$ represents the number of bits needed to represent the modulo value N. Note that Figure 2 shows already the breakdown of the RSA algorithm into the (recursive) modular multiplication blocks (MUL and X^2 for the conventional RSA and MUL_w and X^2_w for the replica).

The circuit takes an input text (either plain or encrypted) and a key (either public or private) and it produces an output text (either encrypted or plain depending on the requested operation or randomly generated in case of an attack has been detected). The conventional RSA takes in input the data to be encrypted/decrypted; The modulo RSA uses the results of a modulo operation of the inputs of the conventional RSA (mod N block).

Both the conventional modular multiplication and the replica modular multiplication are performed in parallel. The *modN* value of the result of the conventional modular multiplication is compared to the output of the replica modular multiplication.

Should this comparison fail, a random output is forwarded to the output of the circuit in order to make differential fault analysis or information leaking unfeasible. To avoid power side-channel attacks, the random number generator is always on.

5. Experimental results

We carried out a series of experiment aimed at assessing the capability of the proposed methodology to protect against DFA and HWT and to measure the introduced overhead in terms of area and power consumption increase and working frequency reduction.

5.1. Experimental Setup

In order to assess the effectiveness of the proposed methodology we considered a real-world HW implementation of the RSA algorithm freely available in the TrustHub repository [17, 36]. To analyse the fault detection capability we implemented a VHDL-level fault simulator that allowed us to emulate the occurrence of bit-flips in both the registers and the output of the conventional RSA circuit. For the HWT detection analysis we considered the four HW Trojans-infested RSA benchmark circuits available in the TrustHub repository [17, 36]. T100 and T300 Trojans leak the secret key by sending it through the primary output of the circuit after a certain number of encyption operations (T300) or when the input plain text is 32'h44444444 (T100); T300 and T400 Trojans are both denial-of-service Trojans activated after a given number of encryption/decryption operations: T300 denies the service by disabling encryption/decryption, while T400 denies the service by

Table 1 Number of required simulations per confidence and error margin value

Confidence Level	99%			95%		
Error Margin	10%	5%	1%	10%	5%	1%
#RSA simulations	638	2551	63756	370	1477	36910

replacing the secret key with a new key only known by the attacker.

All the RSA benchmark circuit implementations have then been extended with the proposed residue checking-based detection mechanism. We used Xilinx Vivado™ [35] as a development and synthesis environment and we then employed the accompanying Device Utilization Summary and Power Estimator tools for area, power and time analysis. In order to have a wide analysis of the efficiency of the proposed technique, we considered two target devices: the Virtex UltraScale+ FPGA (model: xcvu13p-fhga2104-3-e) and the Artix 7 (model: xc7a50tcsg324-1).

5.2. Effectiveness Analysis

Here we first report results referred to the analysis of the fault detection and then to HWT detection.

5.2.1. Fault Attack Detection Capability

We first analysed the effectiveness of the proposed countermeasure in detecting faults in the conventional RSA and the impact of the modulo value on the fault detection capability. When dealing with simulation, as in our case, statistically significant results are usually understood as 99% confidence level at 5% error margin. Based on such a target, the authors in [37] defined the number of experiments, based on the input signals, needed to reach the selected goal. Table 1 shows the number of simulation setups (i.e. different combinations of input values and fault injection location and fault injection clock cycle) required to achieve the desired statistical significance. All circuits (i.e. all modulo values) use the same list of simulation setups. In the remainder of the discussion we report results for 99% confidence and 1% error margin. To do so, we performed about 64000 simulations for each modulo value (which takes roughly 16 minutes to complete in our server when using the batch mode of Vivado™). It is worth mentioning that we randomly chose the circuit's inputs and the FF or wire and the clock cycle where the fault is injected.

It is well known that because of logical and temporal masking, not every injected fault leads to an error

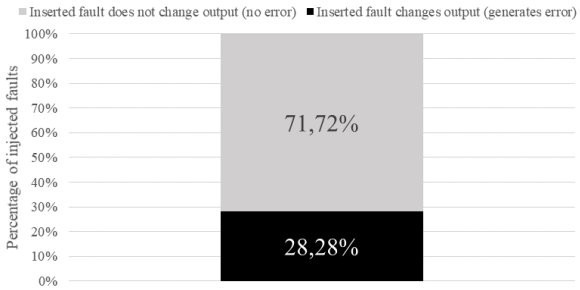

Figure 3: No-effect faults and error-inducing faults

in the output[1]. From a DFA protection point of view the *critical* faults are only those that propagate to the output of the circuit, thus providing information to the attacker. Figure 3 reports the percentage of faults injected in the conventional RSA that modified (and did not modify) circuit's output[2]. It is worth noting that the adopted implementation of the RSA algorithm is already able to tolerate 72% of the injected faults. On the other hand, the remaining 28% of the injected faults are actually able to modify the output of the conventional RSA circuit, thus bringing information to the attacker. For this reason, these faults (now errors) are the ones that we used in the subsequent experiments to assess the effectiveness of the proposed detection technique.

We implemented several versions of the protected RSA circuit with several power of 2 and non-power of 2 modulo values to assess the detection capability w.r.t. the chosen modulo value. We performed a set of fault injection experiments where we injected only the previously identified faults that are able to generate errors in the circuit's output. In each fault injection experiment, a fault was randomly injected in either a flip-flop or an output bit of the conventional RSA at a random clock cycle. Figure 4 reports the percentage of detected errors as a function of the number of bits of the chosen modulo value. For each given number of bits (x-axis), two modulo values have been analyzed: 2^w (grey squares) and

[1] This percentage will be increased by electrical masking when the circuit is working in the field (this cannot be simulated directly with Vivado™)

[2] We did not inject faults in the modulo RSA because this does not bring any advantage to the DFA attacker. As the residue checking will detect also a mismatch in this case.

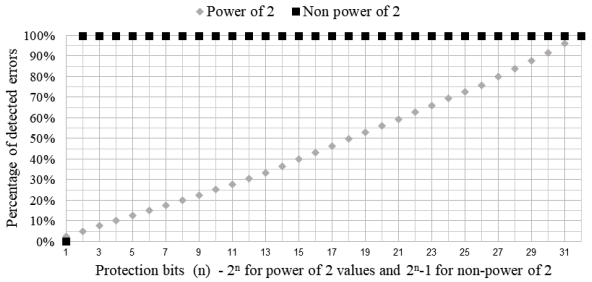

Figure 4: Fault Detection as a function of the modulo value

$2^w - 1$ (black diamonds). It is worth noting that, if the modulo value is a power of 2 then the modulo computation is just a mask operation; otherwise, it requires a full computation with all the bits of the source value (this is actually the advantage of the modulo computation we rely on).

Figure 4 clearly shows that any non-power of two modulo value allows to detect all the errors in the output signals while a conventional power of 2 modulo value is unable to do so. As we previously mentioned, when a power of 2 modulo value is used, only the lower w bits are stored for checking. Thus, any fault in the upper bits is not detected. Consequently, the detection capabilities of power of 2 modulo values are fundamentally linear to the number of considered bits. Obviously, when all the bits are considered (i.e. full replication of the circuit), the percentage of detected errors reaches 100%.

5.2.2. Hardware Trojans Detection Capability

In order to assess the capability of the proposed methodology in detecting the activation of "real-world" HWTs, we extended the four Trojan-infested RSA circuits available on the Trust-Hub benchmark suite (RSA T100, T200, T300 and T400) with the proposed detection mechanism. We chose the $2^{32} - 1$ non power of 2 modulo value to present the worst-case scenario in terms of introduced overhead (discussed in the next subsection). The result of this analysis has been that for all the considered Trojan-infested circuits the proposed technique has been able to detect and signal the activation of the HW Trojan.

5.3. Efficiency Analysis

We measured the area and power consumption increase and the working frequency reduction caused by the proposed detection technique. As a baseline, we considered the hardware implementation of the RSA algorithm available on Trust-Hub.

When implemented on both the considered FPGA devices we did not observe any increase of the number of used flip-flops and I/O pins, while the number of used LUTs increases from 596 to 613 (2.85% more, which, in our opinion, is totally reasonable for a secured system).

Table 2 Power consumption overhead

	UltraScale+		Artix 7	
	Static	Dynamic	Static	Dynamic
RSA	3.067 W	0.018 W	0.070 W	0.0110 W
RSA+	3.067 W	0.021 W	0.070 W	0.0129 W

Table 2 reports the static and dynamic power consumption for the baseline (row *RSA*) and the protected RSA (row *RSA+*) for both the considered FPGA devices. As a first consideration, it can be observed how different are the static power consumption values for the two considered FPGA devices. This is due to the extremely different amount of resources that the two FPGAs have and that the standard compilation is not power gating the unused resources to limit static power consumption. It can also be observed that, in both cases, the proposed protection technique does not affect the static power consumption at all. On the other hand, our solution causes a dynamic power consumption increase of 16.67% and 17.27% in the UltraScale+ and the Artix 7, respectively. Such a significant increase of the dynamic power consumption is mainly due (two-thirds) to the random number generator.

Finally, we analysed the increase of the critical path delay and thus the operating frequency reduction. Both the baseline RSA circuit and the protected one have a maximum operating frequency of 134 MHz on the UltraScale+ device and of about 116 MHz on the Artix 7 device. Therefore, the proposed technique does not reduce at all the operating frequency of the protected system. This result may be explained by considering that the FPGA implementation of both designs is dominated by the path from the input flip-flop, through the multiplier, into the output multiplexor and back to the flip-flop (that stores the partial results). Therefore, the delay observed in the FPGA is mainly caused by the wiring of this path. The modulo RSA and the output check work in parallel with the conventional RSA, therefore, the overall effect is not visible in terms of critical path delay. It is worth pointing out that the lower operting frequency of the Artix 7 and the smaller energy consumption due to the unavailability of CARRY8 blocks, contribute to the lower dynamic power of the Artix 7 implementation against the Ultrascale+ one.

5.4. Comparison

We here report a qualitative comparison of the proposed protection technique against the most recent and relevant related work both for fault attacks protection and for HWT detection.

5.4.1. Fault Attacks Protection

Most of the countermeasures that have been proposed to protect RSA against DFA work at the algorithm level and not at the circuit level. This means that they aim at modifying the algorithm itself or the way it is computed or implemented in order to make the obtained circuit fault-resistant [38, 39, 40]. Therefore, they introduce no (or negligible) area overhead, while the area overhead introduced by the here proposed methodology is 2.85%. Conversely, while algorithm level protection techniques introduce a relevant time overhead (12.50% up to 137.50%), our approach has no impact on circuit's timing since it does not affect the working frequency of the system.

5.4.2. Hardware Trojan Protection

We compared against [41] and [42] which, in our opinion, are the most recent and relevant work related to the proposal in the current paper. Indeed, they both are runtime monitoring techniques working at the system level and they both exploit redundancy. We only provide a qualitative comparison as none of these works evaluate the RSA algorithm. We can observe that in [42] the average area overhead is 95.85% and the average power overhead is 117.40%; no operating frequency reduction is reported. In [41], the introduced area overhead ranges from 15% to 31% with an operating frequency reduction between 1% and 4.20% (no power consumption increase is reported). When compared with these techniques our proposal introduces much smaller area and dynamic power consumption overheads and a no operating frequency reduction. This is of course due to the fact that our methodology is specifically tailored and optimized for the RSA algorithm. On the other hand, our technique cannot be applied to generic circuits as explained in Section 4.1.

6. Security Analysis and Limitations

Here we highlight the limitations of the proposed detection mechanism w.r.t. fault attacks and HWTs. It is worth mentioning that, as for most security countermeasures, we assume that the attacker knows all the details of the adopted solution.

6.1. Multi-bit Fault Attacks

While single-bit fault attacks can be addressed by fault injection (as performed in this paper), multi-bit fault attacks need a careful execution as it is shown in [43] for the AES algorithm. Specifically, the authors of [43] revise the representativeness of published multi-fault attack models and they evaluate the impact of choosing a given model on the results obtained after fault injections.

In this work, we reported single-bit fault attacks. Yet, we have conducted experiments with two-bits fault attacks in the conventional RSA module. The proposed technique is still able to capture 100% of the feasible attacks (for non power of 2 modulo values) and a slightly higher number of attacks for the configurations with a smaller number of bits with power of 2 modulo values. This makes total sense as, for instance, the probability of detecting two bit-flips when just looking at -let's say- 8 bits out of 32 is higher than detecting a single bit flip by looking at 8 bits out of 32. As the number of bits used increases, the detection capabilities converge.

Another threat is the case where the attacker is able to inject faults in both the conventional and the modulo RSA and induce the two modules to generate the same modulo output (so the corrupted value is forwarded to the output). In such case, the proposed protection technique would fail. Yet the occurrence of such an event should be confirmed, at least, through a a realistic modelling of the final circuit and laser capabilities [43]. We performed an initial analytical study of the vulnerability of the proposed solution. First, we concluded that the best place for a multi-bit fault attack would target not only the RSA circuitry but also the final modulo values comparison to make the detection fail (i.e. the produced modulo outputs would be equal and the output multiplexer would forward the corrupted output of the conventional RSA to the output of the system). Table 3 reports the probability of identifying the right bit(s) to flip in one modulo value that -when flipped- match the other modulo value. This probability is the division of the number of combinations that an attack can perform (e.g. "up to 2" means flipping any combination of 0, 1 or 2 bits in the modulo value) by the total number of possibilities (i.e. "up to n", where n is the number of bits of the modulo value). We report results for attacks able to flip up to 2, up to 5, and up to 10 bits; for several modulo value lengths. By looking at the numbers reported in the table, it is clear that the modulo value needs to be long enough to offer enough protection. This is an initial approximation of the vulnerability. The final value will be affected by the distribution of the real values in the inputs, the bits attacked in the RSA, the physical layout

Table 3 Probability of having a match between the outputs of the conventional and modulo RSA modules for several modulo lengths and number of injected bit-flips

Modulo length	#faults injected in the modulo value		
	up to 2	up to 5	up to 10
2 bits	100.00%	100.00%	100.00%
8 bits	14.12%	85.49%	100.00%
16 bits	0.21%	10.50%	89.49%
24 bits	0.00%	0.31%	27.06%
32 bits	0.00%	0.01%	2.51%

of the circuit and the laser capabilities. Yet, this adds another dimension to the study.

6.1.1. Other Attacks

As we have previously discussed, the proposed detection methodology secures RSA HW accelerators against any of the existing DFA attacks except for the safe error attack. Indeed, after detecting a fault (and the experiment demonstrated that the methodology is able to detect all the fault that are actually able to produce an error in the output), the proposed detection methodology will always produce a random output, thus always making differential fault analysis unfeasible. However, the attacker is always provided with the knowledge of whether the fault affected or not the computation. Nevertheless, as it has previously been discussed, the safe error attack is very unlikely to be deployed in real-world, thus it is totally reasonable for a security designer to neglect suck threat. On the other side, the random number generator has to be always on to avoid side-channel power analysis [44]. Indeed, if the random number generator is activated only after a fault has been detected, an attacker able to measure the power consumption of the circuit would observe a power consumption increase and, as a consequence, he/she would discover that the injected fault caused a misbehaviour of the conventional RSA. Such knowledge could then be exploited by the attacker to retrieve information about the functioning of the circuit and the processed data [45].

A much more severe threat for the proposed detection mechanism is related to the output multiplexer that represents a "common-cause of failure". Indeed, by exploiting multiple fault injections the attacker may produce an error in the conventional RSA and then induce the output multiplexer to forward the corrupted value instead of the ad-hoc generated random output. This scenario is not protected by the proposed detection methodology. On the other hand, we argue that this issue may be solved by applying circuit obfuscation/camouflaging techniques to the output multiplexer. Although circuit obfuscation and logic encryption [46] have traditionally been exploited to limit systems' reverse engineering, in the very last years they have also been employed as a solution to prevent DFA, as reported in [47, 48]. The adoption of such techniques would make it hard for the attacker to effectively exploit fault injection in the output multiplexer.

6.2. Hardware Trojans

The proposed technique is intrinsically secure because, as we previously mentioned, the modulo replica is introduced in the system by trusted designers and the fabrication process is also trusted. Since the underlying residue checking mechanism is able to detect any modification of the RSA circuit's outputs the here proposed HWT detection technique ensures the identification of the activation of any HWT able to modify the RSA output signals to either change the circuit's functionality or to steal information. HWTs that do not modify the RSA outputs cannot be detected by the proposed technique: nevertheless, we may argue that, although being extremely stealthy, such HWTs do not represent a threat for the circuit because they are very unlikely to be controlled and exploited by the attacker. HWTs that steal information through covert side-channel and HWT that drain the circuit's battery represent a threat for the RSA circuit, but these HWTs fall outside the threats considered in this paper.

7. Related Work

In the following we review the existing work related to DFA protection against RSA HW accelerators and to detection of HWTs.

7.1. Differential Fault Analysis

DFA demonstrated to be a very effective attack technique. It relies on i) injecting maliciously erroneous values into the cryptographic core while it is performing a number of encryptions/decryptions, ii) collecting the incorrect outputs of the circuit, and then iii) analysing the collected outputs to infer secret information, such as the encryption key. The advantage of DFA consists in letting the attacker select both the erroneous values to inject and the points of the circuit where to inject them. This dramatically reduces the amount of collected data needed to obtain the bits of the secret key, thus allowing the attacker to achieve its goal in a reduced time. It is therefore crucial to protect cryptographic circuits against such attacks in a cost-effective and power-efficient way.

Several approaches have been proposed in the last decades to protect cryptographic circuits against fault injection-based attacks. One approach relies on making the implementation physically inaccessible through tamper-proof boxes and on-chip sensors as for the case of the high-end crypto-core IBM 4764 [49]. This approach demonstrated to be particularly effective but very expensive, since they rely on not-standard technologies. More cost-effective solutions aim at detecting faults instead of preventing them. Duplication/triplication and error detecting codes have thus been employed: these solutions are much cheaper than the previous ones at the cost of a high area occupation and power consumption and longer execution time. Often, these solutions have to be tailored for the specific algorithm under protection.

When looking at protection techniques specific for RSA, one set of works are based on the use of random values during the computation [50, 51]. A different family of works exploit the invertibility of RSA to detect faults during decryption by executing encryption and viceversa [52, 53]. Such techniques protect RSA from attacks based on the observation of the erroneous output. On the other hand, they fail against the so-called *safe error attack*, where the attacker does not exploit the erroneous output but the knowledge of whether the output has been affected by the fault or not [22]. Proposals to tackle such attack have been presented in [54, 55, 56].

A preliminary version of the proposed approach has been applied to protect AES, SHA-2 and RSA from fault attacks in [15, 16].

7.2. Hardware Trojans

The existing HWTs detection techniques may be divided in four main categories: *side-channel analysis-based*, *testing-based*, *proof* and *formal verification-based* and *circuit instrumentation-based*. A comprehensive survey of HWTs detection techniques can be found in [57].

Side-channel analysis-based methodologies [58, 59] aim at detecting HWTs by analysing their footprint in terms of additional power consumption or area occupation, increase of the critical path delay or modification of the surrounding magnetic field. Such techniques are effective for large always-on HWTs, while they may fail in detecting small and triggered HWTs. Moreover, these techniques generally require a trusted golden copy of the fabricated chip, that may be unavailable.

Testing-based techniques [60, 61] aim at exploiting traditional test pattern generation engines to address the detection of HWTs instead of faults. Such techniques are only suitable for triggered change-functionality HWTs, while they cannot be applied to detect always-on and information stealing HWTs. Moreover, as discussed in [62], provided the extremely stealthy nature of HWTs, the generation of test patterns for HWTs detection may be particularly hard or even unfeasible.

Proof and formal verification-based techniques [26, 27] rely on a set of security properties on which the purchaser and the provider of the 3PIP agree before the development of the 3PIP itself. The 3PIP provider agrees with providing a white-box netlist so that the purchaser can verify the security properties before integrating the 3PIP in the system.

Finally, runtime-monitoring based HWT detection techniques rely on additional resources added in the original circuit to provide HWT detection capabilities while the circuit is working. On-chip sensors may be inserted in the circuit to monitor switching activities of internal wires [63], temperature [64] or delays [65]. Hardware redundancy techniques have also been proposed in the last years [41, 42].

8. Conclusions and Future Work

We proposed a lightweight technique for protecting 3PIPs implementing the RSA algorithm against Differential Fault Analysis and HW Trojans horses at runtime. The proposed solution integrates residue checking in the standard implementation of RSA to detect the occurrence of an attack during the execution of the algorithm. Our results show that non-power of 2 modulo values provide effective and efficient implementations to detect both fault attacks and the activation of Trojans. Moreover, we showed that by protecting the RSA 3PIP with the proposed methodology it has been possible to detect 100% of the fault attacks that leaked information to the attacker. Moreover, the proposed technique detected all the Trojans implemented in RSA cores available in the TrustHub benchmark suite. The introduced overhead of the proposed technique is relatively low: we measured a worst case 2.85% area increase, about 18% dynamic power consumption increase and a no impact on the operating frequency.

As a future work, we plan to extend and generalize the proposed methodology in order to be applicable to other cryptographic algorithms and hash functions.

References

[1] J. Katz, A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, Handbook of applied cryptography. CRC press, 1996.

[2] E. Biham and A. Shamir, "Differential fault analysis of secret key cryptosystems," in Annual international cryptology conference, pp. 513–525, Springer, 1997.

[3] M. Joye and M. Tunstall, Fault analysis in cryptography, vol. 147. Springer, 2012.

[4] DIGITIMES, "Trends in the global IC design service market." http://www.digitimes.com/news/a20120313RS400.html?chid=2.

[5] M. Rostami, F. Koushanfar, J. Rajendran, and R. Karri, "Hardware security: Threat models and metrics," in Proc. Int. Conf. Computer-Aided Design, pp. 819–823, 2013.

[6] Mohammad Tehranipoor and Cliff Wang, Introduction to Hardware Security and Trust. Springer-Verlag New York, 2012.

[7] U. Guin, Z. Zhou, and A. Singh, "A novel design-for-security (dfs) architecture to prevent unauthorized ic overproduction," in 2017 IEEE 35th VLSI Test Symposium (VTS), pp. 1–6, 2017.

[8] U. Guin, K. Huang, D. DiMase, J. M. Carulli, M. Tehranipoor, and Y. Makris, "Counterfeit integrated circuits: A rising threat in the global semiconductor supply chain," Proc. IEEE, vol. 102, no. 8, pp. 1207–1228, 2014.

[9] A. P. Donlin, P. Sundararajan, and B. J. New, "Method and system for secure exchange of ip cores," Aug. 2010. US Patent 7,788,502.

[10] M. Tehranipoor and F. Koushanfar, "A survey of hardware trojan taxonomy and detection," IEEE Design & Test of Computers, vol. 27, no. 1, 2010.

[11] X. Zhang and M. Tehranipoor, "Case study: Detecting hardware Trojans in third-party digital IP cores," in Proc. Hardware-Oriented Security and Trust, pp. 67–70, 2011.

[12] J. A. Roy, F. Koushanfar, and I. L. Markov, "Extended abstract: Circuit cad tools as a security threat," in 2008 IEEE International Workshop on Hardware-Oriented Security and Trust, 2008.

[13] G. T. Becker, F. Regazzoni, C. Paar, and W. P. Burleson, "Stealthy dopant-level hardware trojans," in Cryptographic Hardware and Embedded Systems, 2013.

[14] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," Commun. ACM, vol. 21, pp. 120–126, Feb. 1978.

[15] A. Lasheras, R. Canal, E. Rodríguez, and L. Cassano, "Protecting rsa hardware accelerators against differential fault analysis through residue checking," in 2019 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), pp. 1–6, 2019.

[16] A. Lasheras, R. Canal, E. Rodríguez, and L. Cassano, "Lightweight protection of cryptographic hardware accelerators against differential fault analysis," in 2020 IEEE 26th International Symposium on On-Line Testing and Robust System Design (IOLTS), pp. 1–6, 2020.

[17] B. Shakya, T. He, H. Salmani, D. Forte, S. Bhunia, and M. Tehranipoor, "Benchmarking of hardware trojans and maliciously affected circuits," Journal of Hardware and Systems Security, vol. 1, pp. 85–102, Mar 2017.

[18] A. Barenghi, L. Breveglieri, I. Koren, and D. Naccache, "Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures," Proceedings of the IEEE, vol. 100, no. 11, pp. 3056–3076, 2012.

[19] D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the importance of checking cryptographic protocols for faults," in International conference on the theory and applications of cryptographic techniques, pp. 37–51, 1997.

[20] A. K. Lenstra, "Memo on rsa signature generation in the presence of faults," tech. rep., 1996.

[21] F. Bao, R. H. Deng, Y. Han, A. Jeng, A. D. Narasimhalu, and T. Ngair, "Breaking public key cryptosystems on tamper resistant devices in the presence of transient faults," in International Workshop on Security Protocols, pp. 115–124, 1997.

[22] M. Joye and S.-M. Yen, "The montgomery powering ladder," in International Workshop on Cryptographic Hardware and Embedded Systems, pp. 291–302, 2002.

[23] S.-M. Yen and M. Joye, "Checking before output may not be enough against fault-based cryptanalysis," IEEE Transactions on computers, vol. 49, no. 9, pp. 967–970, 2000.

[24] A. Barenghi, G. Bertoni, E. Parrinello, and G. Pelosi, "Low voltage fault attacks on the rsa cryptosystem," in 2009 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC), pp. 23–31, 2009.

[25] R. Karri, J. Rajendran, K. Rosenfeld, and M. Tehranipoor, "Trustworthy hardware: Identifying and classifying hardware trojans," Computer, vol. 43, no. 10, pp. 39–46, 2010.

[26] Y. Jin, X. Guo, R. G. Dutta, M. Bidmeshki, and Y. Makris, "Data secrecy protection through information flow tracking in proof-carrying hardware ip—part i: Framework fundamentals," IEEE Transactions on Information Forensics and Security, vol. 12, pp. 2416–2429, Oct 2017.

[27] M. Rathmair, F. Schupfer, and C. Krieg, "Applied formal methods for hardware trojan detection," in 2014 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 169–172, June 2014.

[28] A. Waksman, M. Suozzo, and S. Sethumadhavan, "Fanci: identification of stealthy malicious logic using boolean functional analysis," in Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, pp. 697–708, 2013.

[29] J. Zhang, F. Yuan, and Q. Xu, "Detrust: Defeating hardware trust verification with stealthy implicitly-triggered hardware trojans," in Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, pp. 153–166, 2014.

[30] Gaisler, "Leon5 processor." https://www.gaisler.com/index.php/products/processors/leon5.

[31] Gaisler, "Noel-v processor." https://www.gaisler.com/index.php/products/processors/noel-v.

[32] J. F. Wakerly, "Principles of self-checking processor design and an example," Tech. Rep. 115, Departments of Electrical Engineering and Computer Science, Stanford University, 1975.

[33] L. Breveglieri, P. Maistri, and I. Koren, "A note on error detection in an rsa architecture by means of residue codes," in 12th IEEE International On-Line Testing Symposium (IOLTS'06), pp. 2 pp.–, 2006.

[34] J. T. Butler and T. Sasao, "Fast hardware computation of x mod z," in 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum, pp. 294–297, May 2011.

[35] Xilinx, "Xilinx vivado design suite - hlx editions 2018.3," dec 2018.

[36] Trust Hub. www.trust-hub.org.

[37] "Chapter 3 - architectural vulnerability analysis," in Architecture Design for Soft Errors (S. Mukherjee, ed.), pp. 79 – 120, Burlington: Morgan Kaufmann, 2008.

[38] C. Giraud, "An rsa implementation resistant to fault attacks and to simple power analysis," IEEE Transactions on Computers, vol. 55, pp. 1116–1120, Sep. 2006.

[39] C. Aumüller, P. Bier, W. Fischer, P. Hofreiter, and J.-P. Seifert, "Fault attacks on rsa with crt: Concrete results and practical countermeasures," in International Workshop on Cryptographic Hardware and Embedded Systems, pp. 260–275, 2002.

[40] J. Blömer, M. Otto, and J.-P. Seifert, "A new crt-rsa algorithm secure against bellcore attacks," in Proceedings of the 10th ACM conference on Computer and communications security, pp. 311–320, 2003.

[41] C. Pilato, K. Wu, S. Garg, R. Karri, and F. Regazzoni, "Tainthls: High-level synthesis for dynamic information flow tracking," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 38, no. 5, pp. 798–808, 2019.

[42] J. J. Rajendran, O. Sinanoglu, and R. Karri, "Building trustwor-

thy systems using untrusted components: A high-level synthesis approach," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 24, pp. 2946–2959, Sep. 2016.
[43] R. Leveugle, A. Chahed, P. Maistri, A. Papadimitriou, D. Hely, and V. Beroulle, "On fault injections for early security evaluation vs. laser-based attacks," in 2016 1st IEEE International Verification and Security Workshop (IVSW), pp. 1–6, 2016.
[44] F. Amiel, K. Villegas, B. Feix, and L. Marcel, "Passive and active combined attacks: Combining fault attacks and side channel analysis," in Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2007), pp. 92–102, 2007.
[45] F. Amiel, K. Villegas, B. Feix, and L. Marcel, "Passive and active combined attacks: Combining fault attacks and side channel analysis," in Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2007), pp. 92–102, 2007.
[46] J. Rajendram and S. Garg, "Logic encryption," in Hardware Protection through Obfuscation (D. Forte, S. Bhunia, and M. M. Tehranipoor, eds.), ch. 3, pp. 71–88, Springer International Publishing, 2017.
[47] J. Howe, A. Khalid, M. Martinoli, F. Regazzoni, and E. Oswald, "Fault attack countermeasures for error samplers in lattice-based cryptography," in 2019 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1–5, 2019.
[48] K. Ngo, E. Dubrova, and M. Moraitis, "Bitstream modification of trivium.," IACR Cryptol. ePrint Arch., vol. 2020, p. 597, 2020.
[49] T. W. Arnold, C. Buscaglia, F. Chan, V. Condorelli, J. Dayka, W. Santiago-Fernandez, N. Hadzic, M. D. Hocker, M. Jordan, T. Morris, et al., "Ibm 4765 cryptographic coprocessor," IBM Journal of Research and Development, vol. 56, no. 1.2, pp. 10–1, 2012.
[50] A. Shamir, "Method and apparatus for protecting public key schemes from timing and fault attacks," Nov. 23 1999. US Patent 5,991,415.
[51] M. Ciet and M. Joye, "Practical fault countermeasures for chinese remaindering based rsa," in Workshop on Fault Diagnosis and Tolerance in Cryptography–FDTC, vol. 5, pp. 124–132, 2005.
[52] M. Joye, "Protecting rsa against fault attacks: The embedding method," in Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC), pp. 41–45, 2009.
[53] A. Boscher, H. Handschuh, and E. Trichina, "Fault resistant rsa signatures: Chinese remaindering in both directions.," IACR Cryptology ePrint Archive, vol. 2010, p. 38, 2010.
[54] C. Giraud, "An rsa implementation resistant to fault attacks and to simple power analysis," IEEE Transactions on computers, vol. 55, no. 9, pp. 1116–1120, 2006.
[55] C. H. Kim and J.-J. Quisquater, "How can we overcome both side channel analysis and fault attacks on rsa-crt?," in Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2007), pp. 21–29, 2007.
[56] S.-M. Yen, S. Kim, S. Lim, and S.-J. Moon, "Rsa speedup with chinese remainder theorem immune against hardware fault cryptanalysis," IEEE Transactions on computers, vol. 52, no. 4, pp. 461–472, 2003.
[57] S. Bhasin and F. Regazzoni, "A survey on hardware trojan detection techniques," in Proc. Int. Symp. on Circuits and Systems, pp. 2021–2024, 2015.
[58] Y. Liu, Y. Zhao, J. He, A. Liu, and R. Xin, "Scca: Side-channel correlation analysis for detecting hardware trojan," in Proc. Int. Conf. Anti-counterfeiting, Security, and Identification, pp. 196–200, 2017.
[59] J. He, Y. Zhao, X. Guo, and Y. Jin, "Hardware trojan detection through chip-free electromagnetic side-channel statistical analysis," IEEE Trans. Very Large Scale Integration (VLSI) Systems, vol. 25, no. 10, pp. 2939–2948, 2017.
[60] S. Dupuis, M. Flottes, G. D. Natale, and B. Rouzeyre, "Protection against hardware trojans with logic testing: Proposed solutions and challenges ahead," IEEE Design & Test of Computers, vol. 35, no. 2, pp. 73–90, 2018.
[61] A. Bazzazi, M. T. M. Shalmani, and A. M. A. Hemmatyar, "Hardware trojan detection based on logical testing," Journal of Electronic Testing, vol. 33, no. 4, pp. 381–395, 2017.
[62] M. L. Flottes, S. Dupuis, P. S. Ba, and B. Rouzeyre, "On the limitations of logic testing for detecting hardware trojans horses," in Proc. Int. Conf. Design Technology of Integrated Systems in Nanoscale Era, pp. 1–5, 2015.
[63] S. Kelly, X. Zhang, M. Tehranipoor, and A. Ferraiuolo, "Detecting hardware trojans using on-chip sensors in an asic design," Journal of Electronic Testing, vol. 31, no. 1, pp. 11–26, 2015.
[64] L. Pyrgas, F. Pirpilidis, A. Panayiotarou, and P. Kitsos, "Thermal sensor based hardware trojan detection in fpgas," in 2017 Euromicro Conference on Digital System Design (DSD), pp. 268–273, IEEE, 2017.
[65] H. Xue and S. Ren, "Hardware trojan detection by timing measurement: Theory and implementation," Microelectronics journal, vol. 77, pp. 16–25, 2018.