Universitá degli Studi di Napoli "Federico II"

Doctoral Thesis

Dottorato in Ingegneria Informatica ed Automatica
XXVII Ciclo

Securing Embedded Digital Systems For In-Field Applications

Author:
Mario Barbareschi

Supervisor:
prof. Antonino Mazzeo
Coordinator:
prof. Franco Garofalo

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy

in the

Seclab Group
Department of Electrical Engineering and Information Technology

March 2015

"Computers are incredibly fast, accurate, and stupid. Humans are incredibly slow, inaccurate and brilliant. Together they are powerful beyond imagination."

Albert Einstein

UNIVERSITÁ DEGLI STUDI DI NAPOLI "FEDERICO II"

Abstract

Department of Electrical Engineering and Information Technology

Doctor of Philosophy

Securing Embedded Digital Systems For In-Field Applications

by Mario BARBARESCHI

Nowadays, special purpose embedded system design relies on the availability of the hardware configurable technology. Space missions, aerospace defense, high performance computing and networking applications benefit from the adoption of field programamble gate arrays (FPGAs) as they provide high degrees of flexibility, fast time-to-market, and low overall non-recurring engineering costs (NRE), but they almost lack in providing security mechanisms to protect intellectual properties (IPs) configured on them. The FPGA programming process is accomplished by a configuration file, so called *bitstream* and hacking attempts can succeed in either cloning the bitstream or, by means of reverse engineering techniques, extracting from it some IPs. Furthermore, through the programming interface, a malicious bitstream can be injected such that the device is reconfigured with a new configuration which overwrites the previous one. The consequences could be really dangerous, not only for the application, but also because they can cause money loss. Since the FPGA programming is pretty much like to software developing process, some existing techniques can be adopted in order to secure the device, mainly involving cryptography primitives. They can guarantee authenticity and confidentiality by exploiting a key stored in each device, but they can be successfully hacked with physical attacks on the device, such that the key is discovered or the configuration file in plain is extracted once deciphered. Recently in the literature, a new technique has been introduced to cope with these issues, called Physically Unclonable Function, since it provides a unique, unclonable and unpredictable hardware fingerprint. Even with the best design effort, PUFs suffer from instability such that their values are variable in time. To face with these issues, this doctoral thesis shows the research activity conducted with the aim of exploring the security threats that characterize the configurable devices and of defining involved roles and new techniques for a design methodology able to guarantee several security attributes, demonstrating the feasibility with a very extended case study, based on a mobile scenario in which high throughput traffic analyzer IP core is distributed to a reconfigurable devices population.

Preface

Some of the research and results described in this Ph.D. thesis has undergone peer review and has been published in, or at the date of this printing is being considered for publication in, academic journals, books, and conferences. In the following I list all the papers developed during my research work as Ph.D. student.

1. Mario Barbareschi, Salvatore Del Prete, Francesco Gargiulo, Antonino Mazzeo, and Carlo Sansone. Decision tree-based multiple classifier systems: an fpga perspective. In *Multiple Classifier Systems*. Springer, 2015

2. Mario Barbareschi, Alessandra De Benedictis, Antonino Mazzeo, and Antonino Vespoli. Providing mobile traffic analysis as-a-service: design of a service-based infrastructure to offer high-accuracy traffic classifiers based on hardware accelerators. *(In press) Journal of Digital Information Management (JDIM)*, 2015

3. Mario Barbareschi, Pierpaolo Bagnasco, and Antonino Mazzeo. Quality trends analysis for the anderson puf varying the supplied voltage. In *Design & Technology of Integrated Systems In Nanoscale Era (DTIS), 2015 10th IEEE International Conference On*. IEEE, 2015

4. Mario Barbareschi, Ermanno Battista, Antonino Mazzeo, and Nicola Mazzocca. Testing 90nm microcontroller sram puf quality. In *Design & Technology of Integrated Systems In Nanoscale Era (DTIS), 2015 10th IEEE International Conference On*. IEEE, 2015

5. Mario Barbareschi, Lionel Torres, and Giorgio Di Natale. Ring oscillators analysis for fpga security purposes, March 2015. URL `http://www.date-conference.com/conference/workshop-w10`. DATE W10 TRUDEVICE 2015, online publication

6. Mario Barbareschi, Antonino Mazzeo, and Pierpaolo Bagnasco. Implementing reliable mechanisms for ip protection on low-end fpga devices, March 2015. URL `http://www.date-conference.com/conference/workshop-w10`. DATE W10 TRUDEVICE 2015, online publication

7. Mario Barbareschi, Antonino Mazzeo, and Antonino Vespoli. Malicious traffic analysis on mobile devices: a hardware solution. *(In press) International Journal of Big Data Intelligence*, 2(2), 2015

8. M. Barbareschi, A. Mazzeo, and A. Vespoli. Un laboratorio elettronico su smartphone per dispositivi a microcontrollore. *Mondo Digitale*, 13(51):207–215, 2014

9. Alessandro Cilardo, Mario Barbareschi, and Antonino Mazzeo. Secure distribution infrastructure for hardware digital contents. *IET Computers & Digital Techniques*, 8(6):300–310, 2014

10. Mario Barbareschi, Ermanno Battista, Nicola Mazzocca, and Sridhar Venkatesan. A hardware accelerator for data classification within the sensing infrastructure. In *Information Reuse and Integration (IRI), 2014 IEEE 15th International Conference on*, pages 400–405. IEEE, 2014

11. Mario Barbareschi, Ermanno Battista, Antonino Mazzeo, and Sridhar Venkatesan. Advancing wsn physical security adopting tpm-based architectures. In *Information Reuse and Integration (IRI), 2014 IEEE 15th International Conference on*, pages 394–399. IEEE, 2014

12. Mario Barbareschi, Alessandra De Benedictis, Antonino Mazzeo, and Antonino Vespoli. Mobile traffic analysis exploiting a cloud infrastructure and hardware accelerators. In *P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2014 Ninth International Conference on*, pages 414–419. IEEE, 2014

13. Mario Barbareschi, Ermanno Battista, Valentina Casola, and Nicola Mazzocca. On the adoption of fpga for protecting cyber physical infrastructures. In *P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2013 Eighth International Conference on*, pages 430–435. IEEE, 2013

14. Mario Barbareschi, Antonino Mazzeo, and Antonino Vespoli. Network traffic analysis using android on a hybrid computing architecture. In *Algorithms and Architectures for Parallel Processing*, pages 141–148. Springer International Publishing, 2013

15. Flora Amato, Mario Barbareschi, Valentina Casola, Antonino Mazzeo, and Sara Romano. Towards automatic generation of hardware classifiers. In *Algorithms and Architectures for Parallel Processing*, pages 125–132. Springer, 2013

16. Flora Amato, Mario Barbareschi, Valentina Casola, and Antonino Mazzeo. An fpga-based smart classifier for decision support systems. In *Intelligent Distributed Computing VII*, pages 289–299. Springer International Publishing, 2014

Contents

List of Figures

List of Tables

Abbreviations

AES	**A**dvanced **E**ncryption **S**tandard
API	**A**pplication **P**rogramming **I**nterface
ASIC	**A**pplication **S**pecific **I**ntegrated **C**ircuit
BOM	**B**ill **O**f **M**aterial
CLB	**C**onfigurable-**L**ogic **B**lock
COTS	**C**omponent-**O**ff-**T**he-**S**helf
DES	**D**ata **E**ncryption **S**tandard
DRM	**D**igital **R**ight **M**anagement
FAR	**F**alse **A**cceptance **R**ate
FRR	**F**alse **R**ejection **R**ate
FPGA	**F**ield **P**rogrammable **G**ate **A**rray
FSM	**F**inite **S**tate **M**achine
HDC	**H**ardware **D**igital **C**ontent
IC	**I**ntegrated **C**ircuit
ICAP	**I**nternal **C**onfiguration **A**ccess **P**ort
IP	**I**ntellectual **P**roperty
JTAG	**J**oint **T**est **A**ction **G**roup
LUT	**L**ook-**U**p **T**able
MRK	**M**aster **R**econfiguration **K**ey
NRE	**N**on-**R**ecurring **E**ngineer
NVM	**N**on-**V**olatile **M**emory
OTP	**O**ne-**T**ime **P**rogrammable
PUF	**P**hysical(-ly) **U**unclonable **F**unction
RO	**R**ight **O**bject in Chapter 3; **R**ing **O**scillator in Chapter 4
SAM	**S**ecure **A**ccess **M**odule

SCA	**S**ide **C**hannel **A**ttack
SDPR	**S**elf **D**ynamic **P**artial **R**econfiguration
SIM	**S**ubscriber **I**dentification **M**odule
SoC	**S**ystem **o**n **C**hip
SoPC	**S**ystem **o**n **P**rogrammable **C**hip
TPM	**T**rusted **P**latform **M**odule
WSN	**W**ireless **S**ensor **N**etwork

Dedicated to the friendship and the memory of Francesco Esposito
(1988 - 2011)

Chapter 1

Introduction

Field-programmable gate arrays (FPGAs), since their first introduction in the 1985, have become increasingly important and essential in the electronic industry. They are today adopted in fielded devices for an extremely wide spectrum of application domains, ranging from consumer electronics to mission-critical equipments, since they are a valid alternative for application specific integrated circuits (ASICs). The cost for a digital system design based on FPGA is lower than the ASIC, although large scale production of the latter surely ensues much cheaper silicon devices. ASICs provide the ultimate in performance, power consumption and number of integrated transistors. Projects based on such technology take very long designing time and expansive costs, with the constraint that, once produced, the digital design that they realize is fixed in silicon and cannot be modified. Meanwhile, developing design changes is much easier adopting the FPGA technology, with also enormous benefits for the time-to-market in digital system projects. Hence, FPGA technology represents a viable solution that is exploitable for small and medium companies which are facilitated in designing innovative solutions and technological novelties in low production scales. In other words, the adoption of FPGAs enables even individual electronic engineers or small teams to realize their digital designs, avoiding huge non-recurring engineering (NRE) costs and the licenses purchasing for expansive synthesis tools used within the ASIC design flow. As indeed, in the silicon manufacturing, to succeed in the consumer market, the production volume for an ASIC has to be enough to divide the NRE costs, as they can run into million dollars, over the produced units to have an acceptable end price for each chip.

Historically, FPGA technology was largely adopted not only in prototyping ASICs, but also as glue-logic between integrated circuits (ICs) hosting small finite state machines (FSMs) and data processing tasks. As they grown in logic capacity and in elaboration speed due to internal structure enhancing, they have been adopting in networking and telecommunication applications, exploiting the huge internal parallelism to meet requirements for large data elaboration. Later on, they gradually have been appreciating by industrial companies which started to adopt the FPGA technology in automotive, processing control and consumer electronics.

Even still so far from ASIC, surely over the time the gap between FPGA and ASIC technologies have been reduced. Modern FPGA technology embeds powerful processing systems, devising the System on Programmable Chip (SoPC) market segment, supporting the developing of systems able to run common operating systems together with custom peripherals, such as Linux, and providing dedicated hardware entities, in hard or soft manner, such as arithmetic and digital signal processing cores, which make easier and quicker the embedded systems designing process [56]. Furthermore, from the early 2000s, they have been equipped with complex and fast communication protocols, able to guarantee high bandwidth and throughput. In combination with classical computer designing, FPGAs can be connected through the peripheral component interconnect (PCI) or Ethernet standards with a complex computer system. Thus a new market segment has been created, called reconfigurable computing, which exploits inherent parallelism degree and reconfigurability of the FPGA technology in order to provide hardware accelerators for software tasks. For instance, crypto-analytic algorithms or simulator engines can take advantage adopting FPGAs to run heavy parallel elaboration processes [62, 51, 30].

Nowadays, FPGAs are practically suitable to design just about anything, directly competing against ASIC technology. Some examples of fielded devices where FPGAs can play a key role include domestic appliances, machine-to-machine devices, such as smart grid sensors and smart meters as well as actuators, for example, smart street lights and smart water heaters, point-of-sale terminals, set-top boxes, personal medical devices and digital media players. Specifically, in automotive industry, advanced driving assistance systems, such as global position system maps, advanced front-lighting system, 3D visualization, collision avoidance system, already make extensive use of FPGA devices. Furthermore, FPGA technology in military and aerospace applications, such as RADAR and SONAR, warfare electronics, unmanned vehicles, rovers and satellite is also expected to contribute to market growth in the next future. As for the consumer electronics market segment, thanks to the growing popularity of smartphones, advanced touch screen functions, phablets, smartwatches, wearable devices, tablets and so on, is expected to drive the demand for FPGA soon. Most, if not all these devices are today connected to some kind of network, forming what is commonly called the 'Internet of Things' [28]. In fact, current estimates indicate that there are some five times more Internet-connected devices than personal computers.

In the above scenario, two fundamental aspects can be easily recognized. On one hand, current fielded electronic systems are much closer in nature to mobile devices than traditional desktop computers. They are essentially embedded systems having various levels of complexity, as they are characterize by severe constraints on computational and energetic resources, most often equipped with a tailored operating system and software environment. On the other hand, the open and connected nature of fielded devices can easily introduce serious security vulnerabilities, treating the applications mission. In particular, while FPGAs provide inherent advantages at the design level, that is, lower time-to-market, easy debugging, near-zero NRE costs, really low bill of material (BOM), rapid prototyping and hardware reconfigurability, they also pose a number of security challenges which might deeply affect the system operation, as they are much closer in nature to the software domain than to mere hardware development.

The FPGA programming is accomplished by providing a configuration file, so called bitstream, downloaded through a configuration interface, mostly using the joint test action group (JTAG) protocol, onto the configuration memory. JTAG is an IEEE standard (1149.1) developed in the 1980s mainly to allow access to the boundary scan chain for test purposes [1], but it has also become a versatile and easy-to-use protocol outside the silicon foundries for giving debug access to IC on boards. Indeed, for the technology continuity, FPGA suppliers adopted such protocol to allow the developers to access the configuration infrastructure. Since JTAG is inherently non secure, because it does not provide neither any access control mechanism nor data confidentiality, the programming interface might be exploit as a backdoor by attackers whose goal is to access both the bitstream and configuration memory.

Malicious users can exploit the JTAG vulnerability reconfiguring the device with a bitstream which is different from the one originally released or reading back the bitstream whereby the FPGA has been programmed. As result, the device could be damaged and, consequently, the application will fail, or some sensitive information can be disclosed, jeopardizing the the user privacy or manufacturer business. Specifically, injecting bitstreams which contain *hardware trojans* [110] in the FPGA, might succeed in causing various types of damages on the device, depending on the privilege granted by the surrounding environment. A bitstream extraction enables to reuse it for other FPGA devices, violating the intellectual property (IP). In fact, the bitstream is produced for a specific device type and it can practically be used to configure all FPGA devices which correspond to such type.

The way in which bitstream is encoded is almost unknown and undocumented, but surely it is not enciphered or confidential in a cryptographic sense, mainly because FPGA vendors want keep this encoding a secret as the bitstream is tightly coupled with the chip internal layout design. As it turns out to be hard, but not impossible, to interpret, in some design projects the bitstream is considered as a secure storage. Reverse engineering techniques can be successfully applied on bitstreams with the aim to extract secrets, e.g. when the application involves cryptography primitives, the bitstream on the FPGA might contain some keys or key materials [25]. Such techniques might be even able to extract and violating single IP cores programmed onto the FPGA.

FPGAs suppliers have taken into account the weaknesses and, little by little, programmable devices and tool-sets are providing some mechanisms which offer security and protection. Furthermore, the research community has given a huge effort in devising new techniques which can be implemented on FPGAs and sometimes cooperating with mechanisms inherently available onto devices. The main exploited security technique is the cryptography: once the bitstream has been output by the synthesis tool, the user can pick a key whereby the software encrypts the sensitive information of the bitstream. The same key has to be programmed into the FPGA device in order to allow the dedicated hard-wired crypto-engine to eventually decrypt ciphered bitstreams before write it in the configuration memory. Furthermore, the FPGA can be programmed in a way which makes impossible the read-back of programmed bitstreams from the internal configuration memory. Malicious users who succeed in obtaining the encrypted bitstream cannot use it since they do not know which key has been used for both configuring the FPGA and enciphering the bitstream. Moreover, they can neither use it on other devices without install on them the same key used to encipher the bitstream nor apply reverse engineer process on its payload.

For the FPGA security design methodology, another technique can deal with IP violation, providing both authenticity and secure key material for cryptographic primitives, based on the Physically Unclonable Function (PUF). In the silicon manufacturing, PUF is the most breakthrough in semiconductor security. The manufacturing process of ICs has unavoidable variations: each circuit design exhibits slightly different electrical behavior from one chip to the next, even though the design, mask and fab are identical. The PUF exploits imperfections that each design circuit instance has in order to output responses which depend on given inputs, namely challenges, providing a challenge/response pairs (CRPs) set as a device fingerprint. Extracted CRPs, among a PUF population, are unclonable, unpredictable, non-reversible, unique, tamper-evident, easy-to-evaluate with a low overhead. PUFs can be successfully adopted to enhance the security of bitstreams, i.e. IP cores, which are distributed among FPGAs [63, 48, 124], even for old families which design project was originally thought without any secure mechanism, exploiting binding mechanism [123], but they cannot guarantee integrity and confidentiality of the bitstream.

1.1 Thesis Contribution

Many factors have to be taken into account when the security of applications deployed on the basis of FPGA technology becomes a crucial aspect that cannot be more postponed. First of all, security is not an absolute attribute and it must be conjugated with other design aspects, such as costs, mission time, time-to-market, power consumption, targeted technology, functional requisites, and so on. Indeed cryptography is not always a viable solution, because needs proper dedicated cryptography mechanisms implemented on chip. Not all the FPGA are equipped with security mechanisms based on cryptography functions and mainly only high-end and modern devices embed them. However, PUFs are easy to integrate in digital design as they are inherently available on any IC, but special attention has to be posed to the quality of the generated CRPs set. PUFs are not stable and their quality has to be proved before being eligible as security primitives. As indeed, the literature has been posing a huge effort in devising high quality and more and more stable PUFs architectures.

Below, with the aim to put in evidence the research activity contribution of this work, there are listed the research questions which I answered:

1. Which are the vulnerabilities that effectively jeopardize fielded applications security which rely on reconfigurable digital devices and, exploiting such weaknesses, which are risks associated to them?
2. Since the FPGA reprogrammability creates the possibility of distributing hardware cores, who are the main actors which are involved in the scenario in which there is an IP core market which distributes them as bitstreams?
3. How the ultimate FPGA technology can guarantee in security, adopting mechanisms inherently embedded by the vendors? How can they be integrated with the remain system, such as the software and operating systems?
4. As for old-families and low-end FPGAs, are PUFs suitable to guarantee minimal security mechanisms? To this aim, which PUFs architectures are available on such a FPGA target?
5. Exploiting reconfiguration and security mechanisms, is it possible to give a comprehensive and complete case-study in which, not only the performance plays a crucial role, but also the security of distributed bitstream is analyzed within the proposed techniques?

As for the last question, the addressed case study is the design of a hardware accelerator for the traffic analysis in the mobile context. As the mobile global traffic will amount to an unexpected value of 24.3 exabytes by 2019 and the smartphone will generate the three-quarters of them, with a speed greater than 2 Mbps [88], and since the number of threats which arrive from the Internet traffic is increasingly, the network traffic analysis is a task that must be integrated within the personal mobile devices. The needing for traffic model updates and the high throughput, in combination with a low-energy consumption requirement, makes the FPGA and a decision tree (DT) based hardware accelerator a very effective solution.

1.2 Manuscript Reminder

This doctoral thesis is structured as follows:

- the Chapter 2 gives an extensive list of vulnerabilities which afflict the FPGA technology, detailing each of them in a technologically manner. Furthermore, some countermeasures are discussed, introducing the main addressed techniques in this doctoral thesis: the encryption of bitstreams and PUFs;
- the Chapter 3 introduces the Digital Right Management architecture to define a secure infrastructure to distributed hardware cores as bitstreams. Indeed, envisioning a real scenario in which devices can partially reconfigure the hardware, the involved roles and the security requirements are discussed in order to discuss and prove the effectiveness of the approach;
- in Chapter 4 two PUFs architectures are illustrated and presented. In particular, the Anderson PUF, which is implemented on a Xilinx Spartan-3E family, and a new PUF architecture, based on the frequencies signature, which is robust against external disturbances;
- the Chapter 5 presents the case study by detailing the methodology exploit to automatically built hardware predictor of trained traffic model, demonstrating the feasibility of the approach. In particular, the hardware accelerator can be securely distributed to devices and it can be updated by collecting new information of traffic from the same devices exploiting an envisioned two-tier service-based architecture;
- At the end, Chapter 6 concludes this doctoral thesis.

Chapter 2

Attacks and Countermeasures Overview of SRAM FPGA Technology

From the research activity background given in the Chapter 1, the FPGA technology emerges as a critical point in guaranteeing in-field applications security. From a layer point of view, such applications can be modeled as in Figure 2.1, in which the hardware layer, represented by the FPGA, hosts some IP cores. Without any trustworthiness guarantee on the underlying hardware, all stacked layers cannot be considered secure. Another problem that arises from the Figure 2.1 is the trust in composition. Given that different design tool-sets produce each layer, or a part of it, and the absence, or the impossibility, of an overarching security architecture, the security associated with the system is as much as least trusted design path.

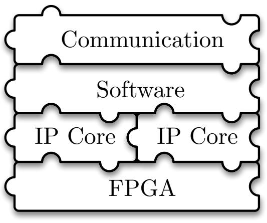


FIGURE 2.1: A simple view of the in-field application layers, which hardware can be configured with some IP cores.

If a critical feature for the security is implemented somewhere in the layers, e.g. the key storage, there is no way to verify that someone cannot tamper or disclose that key. Even with the best approach, i.e. having the control on all the designing processes, cannot assure that the FPGA vendor designed the device without any backdoor or that the silicon foundry did not add some hardware trojans during the manufacturing process. Indeed, being a component-off-the-shelf (COTS), it is right to inquire these questions as much as to raise doubts about the presence of intentional or unintentional design flaws in the IP cores or about the software bugs of involved operating systems which might be exploitable by attacker, and so on.

Indeed, it is possible to affirm that, first of all, in-field applications security relies on the trustworthiness of hardware platforms. The aim of the research activity is not investigate all the non-security sources, hence it is mandatory to start from a trustworthy point in the design chain, otherwise the design level entry has to go back really far, up to the foundry which produces the device or the security of tools involved in the projecting of the FPGA internal design. Hereafter FPGA devices and involved design tool-sets will be considered as the trustworthy level entry. This assumption is not so strong and not so far from the reality, because customers are the first money resources for hardware suppliers and non-secure tools and devices will cause a customer satisfaction degradation, hence a huge money loss for FPGA manufacturer companies. In fact, as the FPGA market segment is really competitive, companies want to have the reputation of being honest, technologically advanced, sellers of secure and high quality devices. These are the most precious assets because are slowly gained by the manufacturer and really easy to lost.

Stated considerations does not imply a greater security level, since, unlikely for fixed and static designed hardware, the FPGA poses a wide range of vulnerabilities because, inherently, the reconfiguration feature implies access to the configuration memory. The only advantage from the security point of view is that, contrary to the ASIC design, sensitive designs are transformed directly by the end-user in hardware components and, consequently, in IP core to configure on the FPGA, without risks of IP theft [116], because no third-parties are involved in the production flow. Figure 2.2 reports a schematic overview of FPGA design flows. The core designer is the IP provider and is able to provide IP cores as HDL projects or as netlist description. The system designer is responsible for the system project, which can be realized with external provided IP, providing as final result the bitstream file. The system manufacturer configures the FPGA with the

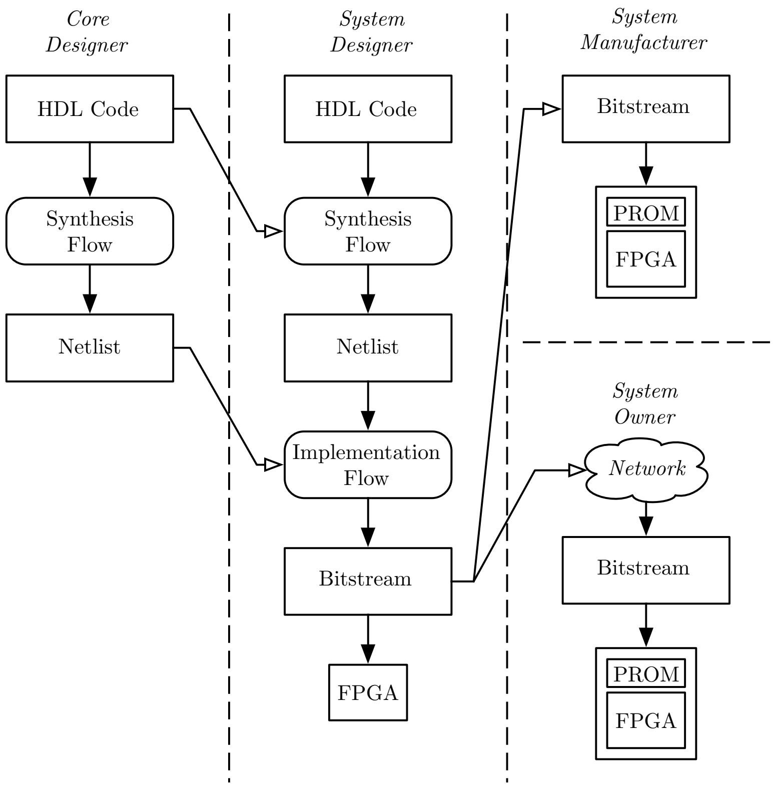


FIGURE 2.2: A schematic overview of roles and project flows which involve in design and deployment phases for FPGA.

obtained bitstream. The system owner is able to reproduce the same mechanism, but doing it directly in-field.

As for third-party IPs, in the industrial globalized economy, the adoption of hardware cores off the shelf is practically a business standard practice. The IP core market is strategic and leads to produce and design an increasingly number of cores, mainly because:

- the design flow is shorter because the reuse strategy guarantee well-tested and dependable components, thus the time-to-market is reduced;
- external IP supplier increases the licensing market, hence profits become higher.

The configurable technology opens more expansion possibilities for the IP marker, since during the device life-cycle many IP cores can be billed and installed on the same device, giving the possibility of maintenance and updates. Moreover, self-dynamic partial reconfiguration (SDPR) interestingly allows to accomplish such updates in an easy and quick way. The FPGA technology that supports SDPR enables the changing of a specific part of the running design, without affecting remain design parts, hence keeping the system properly working.

This new possibility is hard to realize, since requires really high security level which must protect IPs against attacks. Hence, the next Section introduces the main FPGA vulnerabilities which are useful to comprehend the context in which, later in this doctoral thesis, different solutions are discussed.

2.1 FPGA Vulnerabilities

In the digital application design, security techniques constantly evolves. New defense mechanisms are put in place against malicious hacking attacks, but later the system security will be jeopardized by other new attacks that require new countermeasures. In the last years, this security methodology cycle has involved also FPGAs, as they are used in many digital applications that require security features and attackers exploit vulnerabilities to attack such applications. The next Subsections list the most relevant vulnerabilities known in the literature. The first attacks are specifically related to the inner nature of the FPGA and, in particular, to the configurability feature. Then, the Section 2.2 gives an overview of existing countermeasures and the ongoing research on new techniques.

2.1.1 Readback Attack

Most FPGA devices provide debug features, as in the case of ASIC design, via JTAG. Not only this protocol is explored for the configuration of the device, but at the same time it allows the *readback* of the configuration memory of the FPGA. On demand, it is possible to download the current configuration, as a snapshot, from the FPGA. As indeed, the dynamic produced and changed data on LUTs, flip-flops, BRAM, etc., are different from the initial configuration file and can be dynamically gather by reading back the current configuration of the device. Surely, what is read back is not the entire bitstream file, since the snapshot misses the header, footer, no-operation, error detection code and initialization commands.

The idea of the readback attack is simply to read the configuration of the FPGA through the programming interface in order to, by adding what is missing, build the original bitstream file. By applying the readback different times, the attacker is able to observe changes on the FPGA even clock-cycle accurately. Some FPGA families allow to disable the readback functionality providing a security bit. Indeed, more than one bit can be used to disable several features, such as the JTAG boundary access. Mainly, the deactivation of such features is accomplished by some anti-fuses, in order to have a permanent secure configuration once the device is deployed. The idea to disable the boundary access was patented in [9], and described for Xilinx devices in [111]. There are no documented or evince of invasive attacks that re-enabled the readback feature once disabled.

The debugging needs for the readback, so once disabled it is not possible to put the device under test again. Indeed, by properly instrumenting the design, it is possible to extract information about the execution state for specific hardware entities [46]. Xilinx introduced Chipscope suite, which provides hardware probes and the software to debug designs configured on the FPGA [11]. Altera, in the same way, provides SignalTap [115], but apparently without exploiting the readback feature, so inherently the Altera devices are not vulnerable to this kind of attack.

2.1.2 Cloning SRAM FPGA configuration attack

The FPGA technology based on SRAM does not keep the configuration permanently, because when it is disconnected from the power the memory looses the stored data. Hence, at each power-up the device has to be reprogrammed again, by downloading the bitstream from an external non-volatile storage, e.g. PROM or FLASH. An attacker can easily succeed an eavesdrop attack during the programming phase and obtain a copy of the bitstream file.

This attack can be avoided by enciphering the payload of the bitstream with a secret key. As indeed, even if the attacker succeeds in coming into possession of the bitstream, it will be not able to access the original content without having available the key used to encipher it.

2.1.3 Bitstream reverse engineering attack

The bitstream contains the configuration bits that program both the configurable logic and routing paths as established by the user design. Even if this mechanism is pretty clear, the bitstream encoding is practically unknown and without any documentation provided by vendors. Mainly, the way in which vendors encode the bitstream is non-divulged because it is highly coupled with the FPGA internal design and they want to keep it secret. The encoding is an information available only with a signing of non-disclosure agreement. From the user point-of-view this is an advantage because, as it turns out to be very hard to interpret, the bitstream can be considered as a secure storage for design IPs and sensitive information.

However, it is well-known that security-by-obscuring is an approach that is susceptible to affordable attacks. Reverse engineering techniques can be successfully applied on bitstreams to obtain the netlist or something that is similar to the HDL project used to produce it, or to partially reverse it to extract sensitive data, such as keys, from BRAM, LUT or memory cell initialization bits, without having the aim to reproduce full functionality. The full reversal would allow, of course, to manipulate entirely all the designs for a particular FPGA device, obtaining infringed configuration files that are hard to detect.

Two documented reverse engineering experiments are documented in the literature, one performed by NeoCAD and the other by Clear Logic, even if they are not properly examples. Indeed, as reported in [65, 116], they did not reverse the bitstream itself, but they hacked the bitstream generation software tool in order to generate bitstreams able to be configure on FPGAs of Xilinx and Altera respectively.

Anyway, the complexity, the size, the obscurity, the diversity among devices and families that characterize bitstreams really make the reverse engineering task dramatically expansive. As indeed, the literature misses documented reverse engineering successes for modern FPGAs and the legal aspect of the activity is surely a deterrent for such attempts. In fact, the reverse engineering activity might violate the end-user license agreement (EULA) and, consequently, attackers can be sued for damages.

There are some documented partial information extractions, but there is a general lack of automated and general processes, as reported by Ziener et al. in [126]. Furthermore, there was a quite interesting amount of tools and research groups which promised reverse engineering features, but none is now effectively working [89]. Very recently Benz et al. have devised a method to succeed the bitstream information extraction by exploiting a mapping database [25], but the research activity is so far from obtaining interesting and utilizable results.

2.1.4 Side-channels attacks

The nano-scale that characterizes ICs makes them really hard to probe, such that an attacker is not able to retrieve information about data and operations from them. As in the bitstream case, data contained in the IC design are considered securely stored because it is pretty hard to conduct nano-probing attack attempts. This forbids anyone to access IC structures and extract data from it. For long time, cryptographic algorithm relied on this assumption and the involved keys and partial produced data were considered really hard to extract. But so far, Kocher in [61] proposed a technique which is able to break different cryptographic algorithms by correlating the execution time with the key bits. He proposed the first side channels attack (SCA), a class of attacks that extracts on-chip secret data exploiting the external observable phenomena of electronic devices, unintentionally produced by internal operations. The challenge for the designer, which aim is to project devices robust against SCA, is the isolation of internal sensitive operations from their surrounding environment as they are able to interact with off-chip devices with sound, energy consumption variations, electromagnetic and heat radiations [55]. Some SCAs do not require any knowledge on the chip design and are defined as black-box attacks.

As for the FPGA technology, in the literature power analysis attacks [90, 106, 107, 91, 108, 105] and electromagnetic emanation analysis [27, 34] have been addressed.

The power analysis techniques consists in probing the absorbed current, hence the power, during the time. In CMOS technology the main source of power consumption is caused by state changes, 1 to 0 and vice versa, due to the capacitance loading. Collecting the power trace, an attacker can correlate the power consumption with internal operation states, retrieving the targeted data. In [90] a simple power analysis were conducted and, as result, the paper authors succeeded in attacking an elliptic curve implementation on Xilinx Virtex 220 nm device. The authors of [106] addressed the impossibility to attack

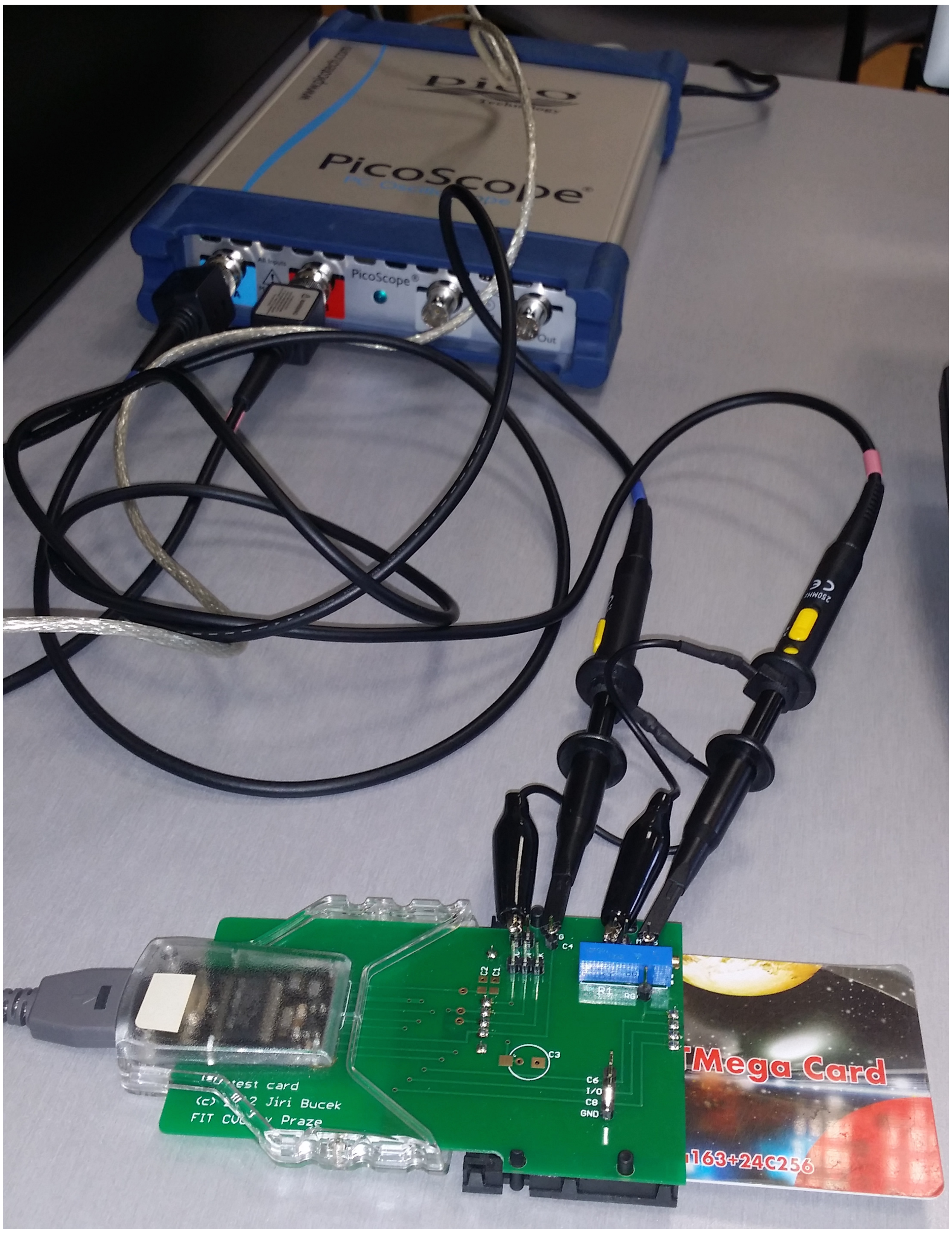


FIGURE 2.3: A side channel attack equipment to perform a power analysis on a smart card and extract the secret key. The attack was performed during the SCA session of TRUDEVICE summer school (Lisboa, July 2014).

paralleled cryptographic architectures with the simple power analysis if other operations are running at the same time on chip. Later, in [107] a differential power attack succeeded on the Advanced Encryption Standard (AES) and Data Encryption Standard (DES). However countermeasures specific for the FPGA have been proposed. One solution has been presented by Tiri et al., which exploits a technique called wave dynamic differential logic [113]. The main idea is to dissipate the power independently from the logic transistor using the differential logic, such that one logic works exactly in opposite from boolean-logic values point-of-view, and the pre-charging of gates. The solution is really expansive, since requires almost doubling logic elements and, consequently, makes the circuit speed lower. Furthermore, since during the manufacturing process ICs are subjected to uncontrolled and unavoidable imperfections, that as explained later in this Chapter (Subsection 2.2.2 are used to realize PUFs), differential logic will always have some observable power variability.

As for the electromagnetic emanation analysis, this technique relies on the production of electromagnetic fields when the circuit moves electric charges during the execution of internal operations. The field can be probed using fine-tuned antennas, amplifying the signal even without removing the chip package. Authors of [27] reported the first electromagnetic emanation SCA on a AES architecture implemented on a 130 nm Altera Cyclone FPGA. Then in [34] an implementation of the elliptic curve crypto-engine on 220 nm Xilinx Virtex have been attacked. Recently, Kim et al., reported in [60] a comparison beetween electromagnetic emanation and power analysis SCA techniques.

2.1.5 Physical attacks

Silicon devices can be physically attacked with invasive probing which aims to extract secret from it. For sure, since the die is packaged, the techniques requires the removal of the package and the treatment of the die in order to access metal layers. Chemical treatments or laser cutter can drill the die, creating some holes for the probes insertion. In order to allow the attacker to accurately observe the die with a microscope and to control the probes and the die position, the physical attack requires a micro-probing station. A more sophisticated technique involves the focused ion beams, which creates incisions at nanometer scale and deposits metal connections, taking high resolution images of the die under attack. Physical attacks on FPGAs are not really feasible, since the shrinking feature size, the IC complexity, the significant number of metal layers, the invasive and destructive nature of probing attacks and the complexity of required tools make them extremely expansive.

Other physical attacks are the semi-invasive attacks, which only require the package removal. In fact it is possible to inspect the die surface by means of imaging techniques or by thermal analysis. Firstly introduced by Skorobogatov in [103], semi-invasive attacks are cheaper than invasive ones, as they do not need an extensive knowledge of the chip. Even though they are feasible to be executed on FPGA devices, no semi-invasive attacks are documented in the scientific public literature.

2.2 Security Countermeasures

Once an attack is disclosed and documented, it is possible to evaluate the vulnerabilities that it exploits and to propose one or more countermeasures against it. First of all, the introduction of new protection is not trivial. Indeed, before the adoption of a countermeasure, the system designer has to evaluate the cost for a security solution together with the probability of an attack, the risk associated with it, required skills to carry it out and the time needed to introduce it in the existing design methodology. Furthermore, some countermeasures imply more than one system modification which could be costly in accomplishing.

As for the FPGA technology, countermeasures can be classified in FPGA manufacturer mechanisms or user-defined techniques. The former represent in-built mechanisms, maybe integrated with the design flow, which FPGA devices provide to end-users. The latter are mainly good practices or secure techniques which can be implemented within the user design. Additionally, countermeasures are further categorized in active and passive. Active techniques are physical or algorithmic deterrents which goal is to prevent dangerous actions attempts of malicious users, such as device tampering or IP and device theft. Passive mechanisms can provide only the proof of intrusion or attacks attempts. In fact, their aim is not to prevent the attack itself, only to recognize it.

In the next subsections, two main techniques are illustrated: bitstream encryption and Physically Unclonable Functions.

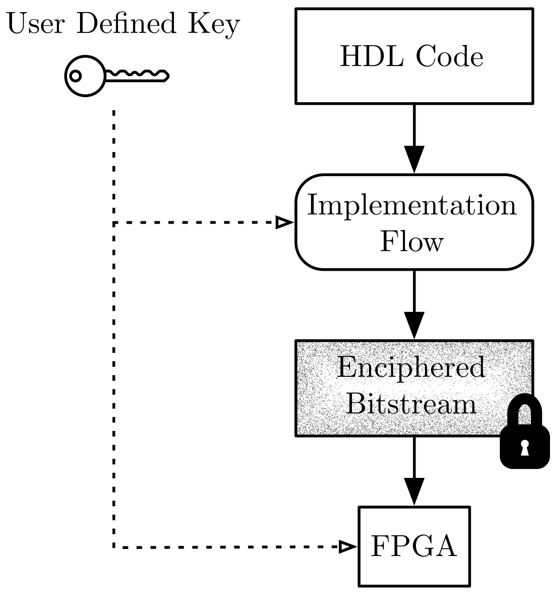


Figure 2.4: Design flow with the encryption mechanisms and user-defined key.

2.2.1 Bitstream encryption

To provide confidentiality to the bitstream sensitive content, an encryption algorithm can be adopted by system designers. Correspondingly, the FPGA must be equipped with a decryption algorithm able to properly decipher the bitstream payload.

Encrypting the bitstream, once produced at the end of the design flow, and decrypting it onto the FPGA forbids the cloning attack, the reverse engineering attack and tampering attempts. The first commercial implementation of the encryption mechanism was in the Actel 60RS SRAM FPGAs family employing a scheme in which, by using the same fixed keys, all the produced bitstreams were encrypted and all the manufactured FPGAs were configured. Hence, this first attempt was only effective in protecting the user design against the reverse engineering, but it did not prevent cloning attacks [58]. Furthermore, succeeding only one single attack enables to decrypt of all bitstreams produced for the Actel 60RS FPGA, making it more attractive for the attackers. Additionally, since the key is fixed and defined inside the synthesis software, rather that attack the device by means of the reverse engineering, attackers might try to attack the software.

The Xilinx Virtex-II family devices were equipped with bitstream decryption build-in mechanism, allowing end-users to configure custom defined keys. Thenceforth, this functionality has been become pretty common, especially for high and middle-end devices. Once the bitstream is completed by the implementation tool, the software looks for an user-define key in order to encipher the payload of the bitstream (Figure 2.4). The user has to install the same key picked in this phase on the FPGA equipped with built-in decryption circuit, before try to configure the ciphered bitstream. When the configuration process starts, the internal logic of the FPGA recognizes, by reading the header data, that the bitstream payload is encrypted and redirects the bit flow to the decryption circuit, which deciphers the stream and pushes it into the configuration memory, as usual. Not only an attacker is not able to perform a reverse engineering attack on the bitstream, but additionally, contrary to the previous implementation devised by Altera, the bitstream cannot be used on other FPGAs which are not configured with the key used during the ciphering operation. Later, Altera introduced the same mechanism starting from the Stratix II family, allowing also the user to force any bitstream trough the decryption engine, such that only ciphered bitstreams are considered as licit, hence the FPGA is protected from the execution of unauthorized bitstreams.

The two previous mechanisms rely on the user-defined key and, in order to thwart its disclosure and consequently nullifying the encryption, it must be kept secret inside the device. One solution is the adoption of volatile on-chip storage, mainly based on low-power memories, which are powered by a backup battery through dedicated pads, such that when the FPGA is disconnected from the power supply the key continues to be properly stored. Another solution is represented by on-chip non-volatile memories (NVM), which permanently store the decryption key. Contrary to the NVM solution, the volatile memory is more resilient to attackers' tampering attempts, since during the physical attacks the battery supply must be always connected to avoid the key loss. But the battery itself requires additional space and a specific holder on board. Nevertheless, system manufacturers have to add the battery cost to the BOM and have to take into account its maintenance during the device lifetime.

Altera Stratix II FPGAs are equipped with a NVM which stores the 128 bit key for the AES decryption, but a special circuitry is needed for the key configuration. With the introduction of the Stratix III family, Altera provides the devices with volatile and non-volatile key registers which can be directly configured through the JTAG protocol. Differently, Xilinx Virtex-II FPGAs have been providing with volatile storage to configure devices with 256 bit key for the 3-DES and users are able to accomplish the key configuration through JTAG. More recently, Xilinx has been starting to provide AES decryption algorithm from the Virtex-4 forward and NVM key register, based on anti-fuse technique, from the Virtex-6 and in all series 7 devices. The anti-fuse technology is one-time programmable (OTP) and allows key program only once, such that during the device lifetime it cannot be changed. When Xilinx devices are configured to work with enciphered bitstream, inherently the readback feature and the Dynamic Partial Reconfiguration (DPR) are disabled. But, using the Internal Configuration Access Port (ICAP) it is still possible to internally read back the configuration bits. Hence, to avoid cloning attack, the other design parts, including the software, have to be secured to avoid access to the ICAP. With the introduction of series 7, the DPR is no more disabled and actually even partial bitstreams can be enciphered and dynamically configured onto the FPGA.

2.2.1.1 Known attacks against bitstream encryption

As stated, without having available the key involved in the bitstream encryption process, an attacker cannot either clone the bitstream or reverse engineering it. The only way to obtain the key is attack the device, by means of SCAs (see Subsection 2.1.4) or physical attacks (see Subsection 2.1.5).

In the literature, there are reported some successful attacks. The first one was performed in 2011 by Moradi et al. [83]. Authors described the attack to the built-in bitstream decryptor of commercial FPGAs, targeting, in particular, a Xilinx Virtex-II Pro device. In the same year, Moradi supposed that the exporting of the SCA technique on other Xilinx families, such as Virtex 4, Virtex 5 and Spartan 6 is not only affordable, but the attack to the built-in decryptor mechanism requires a moderate effort [84]. In 2012 Skorobogatov and Woods performed a pipeline emission analysis to found a backdoor in military grade Actel/Microsemi ProASIC3 FPGA, one of the most secure manufactured FPGA in the market [102]. The most recently attack reported in the literature is by Moradi at al. in [85]. First of all they reverse-engineered the details of the proprietary and unpublished Altera Stratix II bitstream encryption scheme from the Quartus II software and, by using this knowledge, they illustrated a SCA technique that is able to retrieve the full 128-bit AES key in less than three hours.

2.2.2 Physically Unclonable Functions

The Physically Unclonable Function (PUF) is a function that is able to extract secrets that are physically imprinted on a manufactured IC. To this aim, PUFs exploit tolerate manufacturing process imperfections of silicon devices to map a set of inputs, defined as challenges, to a set of outputs, namely responses, forming the challenge/response pairs (CRPs) set. Silicon PUFs have drawn a great attention because they directly exploit the photolithography manufacturing process, hence they do not require additional techniques or specific technologies in order to introduce randomness in the challenge-response behavior. As a matter of fact, programmable devices, such as FPGAs, can be configured with PUFs architectures even if they were not specifically designed with this aim. Since the variability of physical quantities on ICs is an uncertainty source, such that it is unavoidable and uncontrollable, it is unlikely to reproduce two silicon devices with the same defects. Furthermore, the mapping function for the responses generation is hard to predict and, consequently, to simulate. PUFs have a wide range of applications as they can be considered as a hardware fingerprint:secure key generation [3], device authentication [64, 109] and intellectual properties protection [63, 123].

The first idea of PUF was patented by Nanneche et al. in [87], which supposed an authentication mechanism based on the random distribution of ferrite particles over a plastic card. The term PUF was coined 6 years later by Pappu [94]. The first silicon implementation of PUF was illustrated by Gassed et al. in 2002 [44]. They implemented a PUF circuit that exploited delay variations of integrated transistor and wires to extract responses by means of a self oscillating circuit, nowadays addressed as ring oscillator PUF. Another delay based PUF architecture is the Arbiter PUF, which structure is reported in Figure 2.5, introduced in [66], which using two symmetric paths, is able to generate one output bit. Other PUFs architectures can be considered delay based, such as the Anderson PUF, which uses a glitch generated by the difference of delay between two shift registers/multiplexers to preset a flip flop [8]. Kumar et al. introduced in [63] the Butterfly PUF, which exploits two cross-coupled D-latches, and two symmetrical paths to set/reset them.

Completely different, memory based PUFs are based on start-up values of memory cells, which are unpredictable, unclonable, easy to evaluate and, mainly, they are technologically available on a wide range of ICs, hence do not require additional resources. In [49]

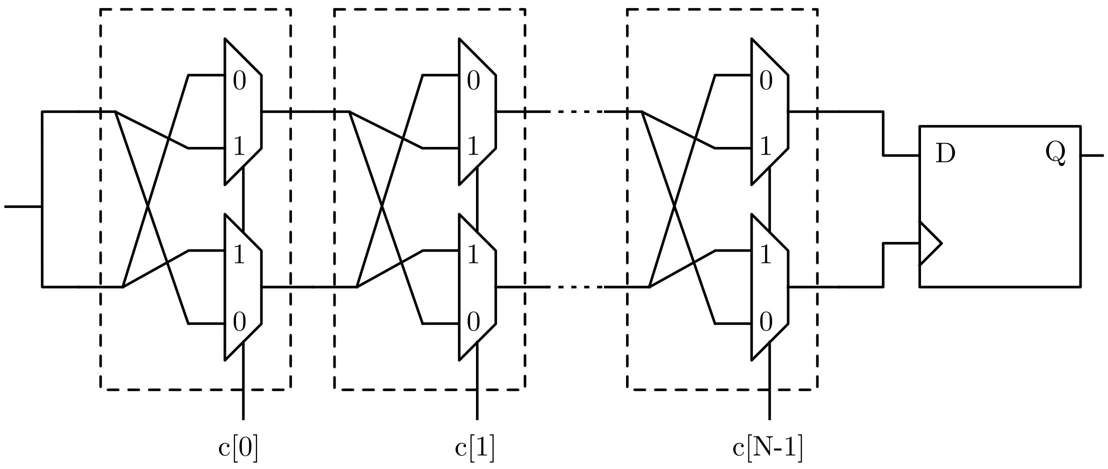


FIGURE 2.5: The Arbiter PUF exploit one symmetric path, selecting it by means of multiplexers, in order to extract one response bit.

this concept was exploited for FPGAs which embed SRAM blocks, but also other memory types can be exploited for the aim, such as the D flip-flop PUFs, which basically exploit the same mechanisms of the SRAM puf, but using the D flip-flop of FPGAs [71]. Recently, a PUF design based on the Spin-Transfer-Torque Magnetic memories (STT-MRAMs) has been introduced in [33] and [118].

PUFs can be further categorized in weak and strong. The former are PUFs with a small (or even singleton) CRPs set, such that can be only adopted as key storage or material. The latter are PUFs which CRPs set cardinality is really huge and they can be also adopted in authentication protocols. A simple authentication protocol can be implemented as reported in Figure 2.6. A trusted-third party (TTP) randomly extracts a subset of CRPs set from a device which embeds a PUF before deploy it. This subset is stored in a secure database, completing what is called PUF *enrollment phase.* When the TTP has to remotely authenticate a deployed device, it picks a challenge from the stored subset and queries the device PUF in order to receive a response. If the received response is equal to the stored one, the device is authenticated. Since challenges and responses are exchanged in plain, each used challenge will not be used again in order to avoid the man-in-the-middle attack.

PUFs are able to successfully substitute volatile or non-volatile memories to store cryptographic keys. Indeed, instead of storing the key in a register, the cryptographic algorithm can directly use as key a PUF response. Such architecture is addressed as *hardware entangled cryptography* [70]. The advantages are the uniqueness of PUFs generated keys and the inherently protection of the challenge/response mechanism. Indeed, contrary

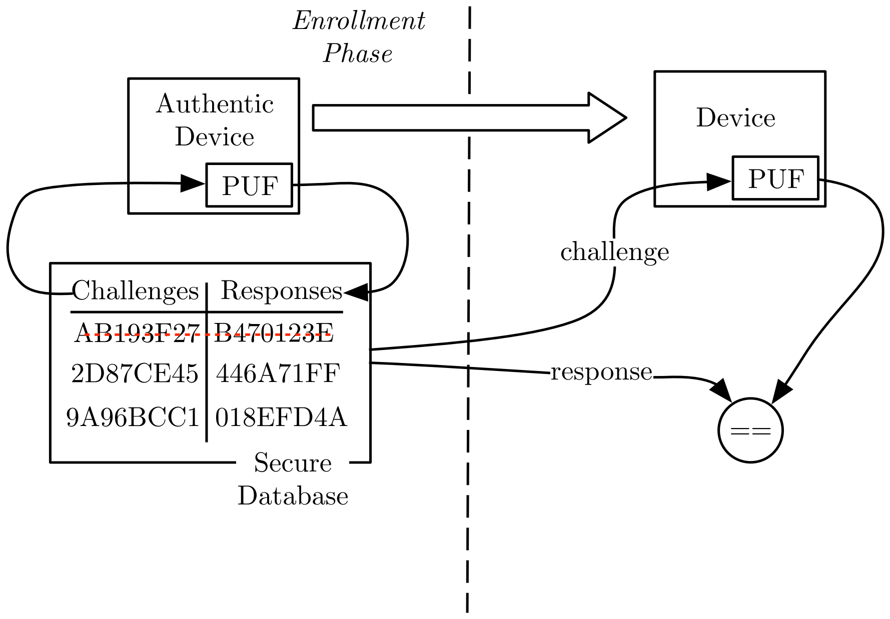


FIGURE 2.6: Simple authentication protocol based on the challenge-response mechanism of PUFs.

to memory approaches, PUFs give a response only when the circuit is powered-up and a challenge is provided. Furthermore, the nature of electrical phenomena makes the PUFs tamper evident circuit, such that a physical attack against a device permanently changes the responses of the PUF which is embedded on it.

The silicon PUFs responses are not stable during the time, hence they cannot be consider as functions in mathematical manner. PUFs exploit electrical phenomena, mainly the signal propagation delay, as secrecy source, and they are susceptible to uncontrollable working conditions, such as supply voltage fluctuations, temperature variations, electro-magnetic fields interactions, device aging, and so on, producing noisy responses in time. This implies that each challenge might be mapped into more than one response. Fuzzy extractor techniques can be successfully adopted to recover noisy responses. It involves the evaluation of the PUF stability in order to extract some data, namely helper data, which can be used to provide noise-free responses. Since the helper data cannot be used by attacker to guess the PUFs responses, it is not critical and can be even stored in non-secure memories [37].

2.2.2.1 PUFs properties

This subsection illustrates a formal perspective of the PUF, mainly to better understand properties which characterize it. First of all let the PUF be a mathematical application which associate challenges to responses. Ideally such application is a function:

$$\theta \in \Theta : C \to R | \theta(c) = r, c \in C, r \in R. \tag{2.1}$$

C is defined as the allowable challenges set and R is the responses one. Θ includes all the PUFs population in order to identify silicon devices which embed them. It is worth noting that, in real cases, the challenge-response relation is substituted by $\theta(c) \approx r$. From the probability point of view, r is the most probable response that can be output from the PUF by stimulating it with the challenge c.

Unclonability The main PUF property is the unclonability and can be formally described as the impossibility to obtain a PUF $\widehat{\theta}$ which is identical to θ, such that the latter can replace the former. Being a physical object, the unclonability can be considered in a mathematical or physical manner. Mathematical unclonability implies that it is hard to find a mathematical procedure that is able to provide the same CRPs set:

$$\nexists f : \omega(c) = f(c) = r, \forall c \in C. \tag{2.2a}$$

Physical unclonability indicates that it is hard to reproduce a PUF which can be recognized as another one:

$$\nexists \sigma : \theta(c) = \sigma(c) = r, \forall c \in C. \tag{2.2b}$$

Uniqueness The PUF uniqueness property is formulated as:

$$\nexists \widetilde{\theta} \in \Theta : \widetilde{\theta}(c) = \theta(c) = r, \forall c \in \widehat{C} \subseteq C. \tag{2.3}$$

The definition can be interpreted in ambiguous way since it seems to be overlapped or very similar to the uniqueness statement (Equation 2.2b). However this is not true, because the requirement to be unique for a θ is the non-existence of another $\widetilde{\theta}$ that specifically belongs to Θ, unlike the unclonability that requires the non-existence of a generic function that is able to substitute a given PUF. Moreover, the uniqueness requires to be valid only for a subset of C.

Apart the theoretical meaning, the uniqueness has a practical usefulness. Indeed, let $\dot{c} \in C$ be a random picked value and $\dot{r} = \theta(\dot{c})$; the pair $(\dot{c}, \theta(\dot{c}))$ induces a partition into Θ such that $\dot{\Theta} \cup \bar{\Theta} = \Theta$ and $\dot{\theta}(\dot{c}) = \dot{r}, \forall \dot{\theta} \in \dot{\Theta}$ and consequently $\bar{\theta}(\dot{c}) \neq \dot{r}, \forall \bar{\theta} \in \bar{\Theta}$. A successive picking of $\ddot{c} \in C$ define another pair $(\ddot{c}, \theta(\ddot{c}))$ such that causes a partition in $\dot{\Theta}$, and so on. By nesting this approach, so successive applications of other c values on θ, the set $\Psi = \{(\dot{c}, \theta(\dot{c}))\}$ will cause increasingly smaller partitions, up to having a singleton set in which only one function, θ, fulfills all the conditions. The success of this procedure and the required steps to complete it depend on the Θ cardinality, the characteristics of the PUF θ and of the picked c.

Unpredictability The unpredictability property can be directly inherited from the definition of mathematical unclonability, Equation 2.2a.

$$\Psi = \{(c, \theta(c))\}, \nexists \Phi : \Phi(\Psi, c_p) = \theta(c_p) = r_p, \forall c_p \in C. \tag{2.4}$$

What is requested to be unpredictable for a function is the inability to create a procedure Φ that, having a certain amount of challenge-response pairs Ψ for a PUF θ, is able to provide the same output of θ for a generic challenge c. The existence of this procedure is in directly contrast with the unclonability because Φ represents a mathematical clone that can predict the θ responses.

One-way property Formally θ is a one-way function $\iff$ given $r = \theta(c)$ it is hard to find $\lambda : \lambda(r) = \bar{c}$ and $\theta(\bar{c}) = r, \forall \bar{c} \in \bar{C} \subseteq C$. As for the hash functions, in this definition "hard" is meant in the computational theory sense, so that given one output r of a PUF θ, it is very expansive to find one input $\bar{c}$ such that $\theta(\bar{c}) = r$.

Evaluability Given $\theta \in \Theta$ and $c \in C$, θ is evaluable if it is not hard to evaluate $\theta(c)$.

Tamper-evident θ has the tamper-evident property $\iff$ any attempt to tamper the PUF θ permanently changes it in $\theta' : \theta'(c) \not\approx \theta(c), \forall c \in C$.

2.2.3 Known attacks against PUFs

Not all the proposed PUFs architectures have the previously listed properties and some of them are partially proved. Recently, some PUFs have been discovered susceptible of attacks. Surely, these attacks are specialized for a specific PUF design and cannot be universally applied to every PUF.

Model-based attack Some PUFs are susceptible of model based attack. In particular, strong PUFs, i.e. PUFs with a large CRPs set, can be attacked by means of machine learning algorithms. If their interface is public, as in the authentication scheme reported in Figure 2.6, and can be easily accessed, it is possible to obtain some CRPs. Model based attacks require a huge amount of CRPs during the learning phase and, consequently, for the strong PUF it is not a hard-to-meet constraint. Once extracted, the data model is able to give predictions with an high accuracy rate. In [96] Ruhrmair et al. have demonstrated this approach attacking some implementations: RO PUF, arbiter PUF and arbiter variants. Trained models were able to predict a response for a given challenge with an accuracy rate near to 99%. Another interesting result has been reported in [73] by Mahmoud et al.: combining machine-learning techniques with leak information through side channel, authors have attacked a xor arbiter PUFs with challenge width of 64, 128 and 256 bits.

Weak PUFs cannot be a target of such attacks, since they cannot be provide a CPRs set huge enough to train a useful model.

SCA As previously introduced in Subsection 2.1.4, SCA analyzes with statistical tools secondary effects produced by ICs when they are working, especially targeting cryptographic algorithms. Even if the PUF itself is SCA-resistant, post-processing techniques can jeopardizes the secrecy as they might be weak against SCAs. The first SCA attack on post-processing algorithm was performed in [57] by Karakoyunlu and Sunar. The complexity of the fuzzy extractor obligates the system designer to implement it through a software algorithm, that is easy to break by means of SCA.

Exploiting electromagnetic emission of ROs, even very small, some papers reported attacks that can be used to guess the ROPUF responses [79, 78, 80]. At the same time, they have provided some countermeasures to avoid such SCAs.

Chapter 3

Secure Infrastructure for Hardware Digital Content Distribution

The FPGA security is a fundamental aspect for in-field applications, which exploit such reconfigurable technology not only for immediate accessibility in the market, but also for inherently features that define new design paradigms. In particular, the reconfiguration ability is able to provide an interesting adaptability and flexibility to the functionality and computation required when, some point in time, the current configuration is inadequate. Being in-field, such feature has to be remotely exploited by transmitting the new configuration. More largely, this scenario can be extended considering a market in which device reprogrammability creates the possibility of distributing hardware cores, pretty much like software digital contents. The hardware update mechanism must guarantee the respect of the IPs rights associated with it.

To effectively extend the above scenario, since the hardware update is so close to the software case, it is mandatory to explore existing techniques for the software domain, identifying involved roles and mutual relations among them. In the next Section, some proposals from the research literature are given, then the Digital Right Management (DRM) technology is briefly discussed in order to list and detail who are the parties involved in the envisioned scenario.

3.1 Existing Research Proposals

Several proposals in the literature protect the IP of full and partial bitstreams by using encryption schemes. In [52], the authors implemented on a Virtex5 a partial reconfiguration module exploiting AES-GCM in order to implement encryption and authentication. Some performance analysis are provided. In [26], the authors proposed an efficient Authenticated Encryption scheme tailored for FPGA bitstream protection. It is based on the AES-based authenticated encryption scheme. The authors of [36] focused on spoofing and reply attacks. With spoofing, an attacker can replace the data with fake information (see Subsection 2.1.2). With a reply attack, an attacker can record the bitstream sent by the user to the FPGA and re-send it in order to delete subsequent configurations. The authors proposed a frame tailored for trusted FPGAs or trusted boards whereby they guarantee confidentiality, integrity, authenticity and up-to-datedness based on AES and Hashed Message Authentication Code (HMAC) algorithms. The authors of [45] proposes a technique to make Xilinx low-cost FPGAs as secure as high-end FPGAs without adopting an encryption scheme. They use one or more PUF (see Subsection 2.2.2) to generate a signature used by a host to obfuscate the hardware description language (HDL) code, such as a finite state machine (FSM) state coding. The generated bitstream is well-functioning only for a specified device. In [59], the authors illustrated a methodology to defend an IP by using trusted platforms based on FPGA dynamic reconfiguration capabilities. They identified the players, such as the system integrator, trusted authority, FPGA fabric vendor and so on, and defined a few solutions to add, update, store and catalog an IP. They provided also a proof-of-concept implementation on a Virtex-II FPGA, but no details about the protocols were given. Couture and Kent [31] extended a licensing approach to hardware components. The authors concluded that this approach requires a modified FPGA architecture equipped with a secure NVM and a tamper-resistant unique identifier. Güneysu et al. in [50] proposed a volume licensing scheme for FPGA bitstreams. The proposal requires changes to the FPGA configuration controller, since a secondary key register is needed in addition to the built-in encryption mechanism. Furthermore, the solution is limited to the protection of full FPGA configuration, so it is not possible to protect partial soft IP cores. Drimer et al. extended this work for the protection of multiple cores [38]. Maes et al. in [72] proposed an IP protection mechanism for FPGA designs at the level of individual IP cores. In fact, although the reseach activity of this doctoral thesis was developed independently, the proposal in [72] is closely related to the solution which is here proposed. Indeed, similarly to the work presented here, they relied on self-reconfiguring capabilities, they only used primitives already available in existing FPGA devices, and required an external TTP. This makes it possible to enforce a pay-per-use licensing scheme. Unlike the work presented here, however, they did not exploit built-in mechanisms for partial bitstream encryption, hence requiring a configuration bitstream with a custom AES engine to be expressly installed upon each partial configuration. In addition, the metering service is managed online by the TTP, unlike the concept of RO used here, borrowed from the DRM domain, letting the end used autonomously consume hardware contents. Furthermore, Maes et al. did not demonstrate any practical implementation, particularly concerning the connection with the software layers and the possible implications in terms of software DRM [72]. At the end, Thanh et. al in [112] gave an overview on techniques and involved services for an IP protection infrastructure, but without exploiting AES decryption inherently installed on the FPGA.

3.2 Digital Right Management

DRM refers to the protection, distribution, modification and enforcement of the rights associated with the use of digital contents [39, 95]. Figure 3.1 shows the classical architecture of a DRM infrastructure in the software domain. The content provider (the party who has created the content) packages data according to a DRM specification and establishes one or more sets of usage rights and related usage costs. Consumers can buy and download contents and associated rights from third-party sources, for example, from the website of a content distributor. In this scenario, a client DRM-enabled platform guarantees that a proper license has been obtained by the user and enforces rules for playing contents. All DRM mechanisms rely on some form of specialized hardware to achieve secure delivery of contents, usage rights and authentication. Indeed, since user identification, cryptographic capabilities and the ability to resist against external tampering are vital to DRM systems, implementations of DRM are often based on trusted computing platforms [47], which provide the above benefits as a native feature. Such platforms mount an additional chip, the Trusted Platform Module (TPM), similar to a smartcard in its internal structure, providing security functions such as platform attestation, protected storage and sealing, to measure and validate the hardware and/or

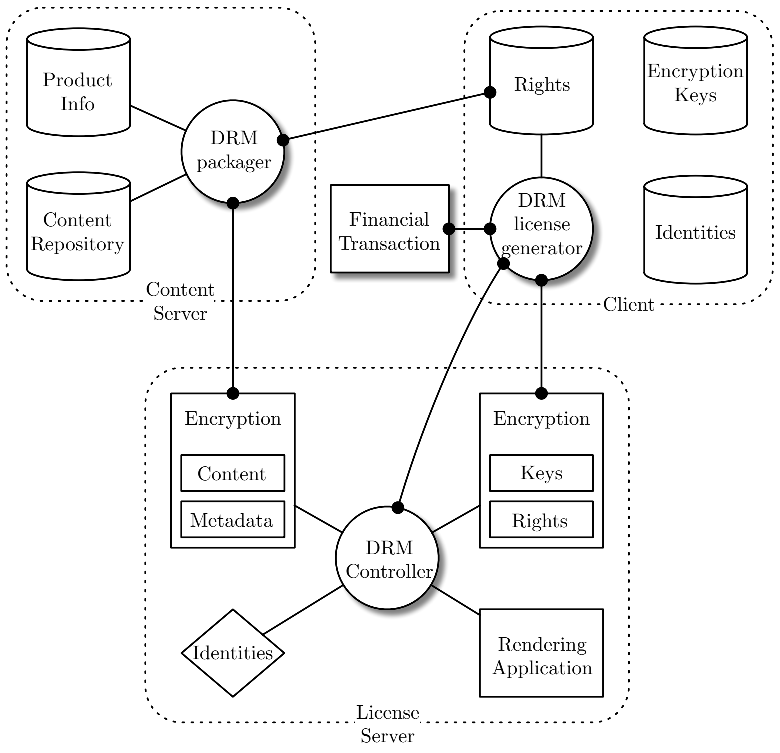


Figure 3.1: Schematic of the classical architecture of a DRM infrastructure in the software domain.

software configurations of the platform [47]. In particular, the Open Mobile Alliance (OMA) DRM specification [4] provides a reference architecture for DRM in mobile and handheld devices. Based on trusted computing facilities, the OMA DRM architecture specifies the scheme of Figure 3.1 by defining four actors that interact with each other in order to distribute protected digital contents to end-users. The content issuer, as the owner of the digital content, firstly converts the content to an appropriate format and then negotiates a right object (RO) with one or more right issuers (RI). The RO describes constraints and permissions granted to the DRM Agent for a specific content. Before selling an RO to the end-user, the RI sets up a trusted relationship with the DRM agent, enabling such features as mutual authentication and encryption needed to distribute copyrighted content. In OMA DRM specification, these trust relationships are based on public-key infrastructure certificates issued by a third-party trusted authority, attesting both users and RI identity.

DRM plays an important role in current mobile environments. For example, Android 3.0 [35] and higher platforms provide an extensible DRM framework that lets applications manage protected contents using a variety of DRM mechanisms. The DRM framework offers an abstract, unified Application Programming Interface (API) hiding the complexity of content management operations. The architecture of the framework is plugin-based, allowing device manufacturers, content owners and Internet digital media providers to easily implement and extend the mechanisms, enforcing content protection through Android applications. A prominent example of an extension is the Widevine third-party plugin [2], built on top of the Android DRM framework, offering advanced DRM and copy protection features. Protected content is secured using an encryption scheme based on the open AES algorithm. An application can decrypt the content only if it obtains a license from the Widevine DRM licensing server for the current user. The Widevine DRM plugin integrates with the hardware platform to leverage the available security capabilities. The level of security offered is determined by a combination of the security capabilities of the hardware platform and the integration with Android and the Widevine DRM plugin. In fact, Widevine DRM security supports three levels of security [2], essentially depending on the portion of the underlying system that is physically secured from hacking (e.g. DRM master keys, system boot, content rendering hardware etc.).

3.2.1 Roles in hardware-level DRM

Device reprogrammability creates the possibility of distributing hardware cores, for example, video codecs, pretty much like software digital contents, possibly on payment or on a subscription basis. In fact, as part of the application logic, hardware accelerators can be perceived as add-ons that can be downloaded, possibly on payment, and installed on the user's device to extend its processing capabilities. This resembles very closely the scenario where software digital contents, for example, tones, MP3 files or mobile application themselves, are distributed through a controlled licensing service.

With the aim to extend the notion of digital contents to reconfigurable hardware components, called here hardware digital contents (HDCs), it is necessary to propose an infrastructure for the secure distribution of such HDCs and a device architecture which can implement secure mechanisms required by DRM. In particular, this Subsection defines the roles involved in the distribution infrastructure, followed by a brief analysis of the main security threats and the essential functions that the hardware solution must offer.

End user The first identifiable role is the owner of the device, namely the End User. The End User's device contains a hardware-reconfigurable subsystem, which can be possibly updated in the field. The End User requires the device to be secure, in order to protect not only the personal data, but also to keep the system properly working. On the other hand, the other parties want to protect themselves against the End User to avoid the cloning of the equipment (e.g. by copying the FPGA bitstreams or partial ones) or the unauthorized replacement of the FPGA design. For instance, in the case of pay-per-use HDCs delivery system, the End User might reconfigure the system without effectively complete any payment transaction. The End User also is a part of the licensing process when it becomes the owner of the system and wants to bill new HDCs.

FPGA Fabric Vendor The manufacturer of the FPGA chip within the device is identified as FPGA Fabric Vendor. It must guarantee some basic secure mechanisms in order to provide integrity and confidentiality of both static and partial bitstreams. Its devices should not be equipped with backdoors, because the FPGA Fabric Vendor has to respect the intellectual property rights and avoid the usage by an attacker.

IPCore Vendor The design and the implementation of the intellectual property as HDC is done by the IPCore Vendor. The HDCs are available as partial bitstreams to be downloaded to the FPGA. The IPCore Vendor requires the protection of its own IP rights, not only against the overbilling mechanism or unauthorized distribution, but also to keep the design confidential. The IPCore Vendor is aware of the requirements that it has to fulfill, such as the compliance with the IP design specification and with the End User system. The pay-per-use mechanism is an IPCore Vendor's concern, meaning that the issued HDC has the responsibility to provide a cryptographic key specific to the HDC.

Trusted-Third Party The TTP, or Trusted External Party, is an organization that all parties rely on. The infrastructure guarantees that all the transactions among the other parties are securely completed. For this reason, the TTP stores in a registry all the parties' keys, as well as the correspondence between the FPGA-ID and the installed secret key. The TTP needs also to install the first license record when the End User bills the HDC.

3.2.2 Attacks scenario

In Section 2.1 FPGA vulnerabilities have been listed and explained in detail. Here they are considered as applied to the envisioned scenario, in which (partial) bitstreams are distributed as HDC with digital rights.

Malicious bitstreams Third-party bitstreams are inserted into the hardware environment to either damage the device or steal sensitive information. In particular, HDC bitstreams might contain hardware Trojans that, depending on the privilege levels granted by the surrounding environment, might cause various types of damages on the device.

IP theft and reverse engineering of bitstreams An attacker accesses the plain bitstream in order to realize compatible, unprotected clones of the distributed components. The original bitstream can be obtained by adding a proper header and other configuration structures which are missing in the stolen bitstream (see Subsections 2.1.2 and 2.1.3).

Digital right tampering The end user manipulates the ROs associated with a component downloaded to the device in order to obtain unauthorized accesses. This might be done, for instance, by hacking the digital right representation in an NVM, for example, by changing the value of a counter which keeps track of the remaining number of uses.

Software tampering Since no assumption is made in this work on the security measures adopted by a software environment (in particular, here there are no assumptions about the environment, which is not necessarily compliant with a trusted computing architecture) an attacker might tamper with the software libraries interacting with the reconfigurable hardware component. In particular, the software part coming with the HDC might be completely hacked, while – in principle – the HDC might be emulated in software by an unauthorized library injected by the attacker.

3.2.3 DRM hardware functions

In order to effectively tackle the above security threats, some vital functions have to be supported by both the secure HDC distribution infrastructure and the corresponding device architecture.

HDC validation HDCs are very low-level in nature, since they involve the hardware part of the system. While the APIs ensures some form of isolation from the static part of the FPGA and the software environment, the open nature of the platform might introduce serious vulnerabilities exploited by malicious HDC developers. It would thus be highly desirable that the distribution infrastructure offers an off-line validation service. Given the reduced computing power normally available on the end user device, it is in fact unlikely that an HDC check takes place on the device itself, like an antivirus scan on a standard general purpose platform. Ideally, the HDC should undergo some form of verification at the time of registration by means of an external player, which is also in charge of HDC distribution and secure packaging.

HDC distribution and billing While validation guarantees the security from the user's perspective, the infrastructure should also offer a platform for the protection of the Intellectual Property to third-party developers, ensuring secure distribution and billing of HDCs. The infrastructure should enable various business models, including pay-per-use policies for the downloaded components, preventing the uncontrolled redistribution of HDCs.

HDC digital rights Each distributed HDC will have a set of digital rights associated with it. These constitute the essential cryptographic element enforcing the distribution policies requested by the third-party developers, for example, per-use licensing, duration-based licensing and so on.

3.3 Hardware Architecture

With the goal to realize a working prototype of the device that can works within a DRM architecture, the FPGA has to be picked among commercially available programmable devices. This results in a number of technological constraints that were taken into account for the definition of the device architecture and the main interactions taking place for HDC distribution. The technological constraints directly reflect the security-related features of existing FPGA families [104]. First, a secure configuration mode should be available, where the configuration bitstream is stored outside the FPGA device in an encrypted form and is only decrypted inside the FPGA within a secure perimeter. Readback on the unencrypted bitstream is inhibited. Symmetric encryption (e.g. AES encryption) is the only mechanism available as the built-in mechanism for securing the FPGA bitstream, and a single symmetric key for enciphering both static and partial bitstreams is available. Furthermore, no NVM is available on-chip in the hardware execution environment, that is, the FPGA device. Concerning the interaction with the End User's host device, HDCs are accommodated on a partially reconfigurable part of the FPGA (whose perimeter is defined statically at design time) although they are provided with an API allowing them to interact with both the static part and a software environment running outside the reconfigurable fabric.

The device architecture is shown in Figure 3.2. The hardware execution environment contains a partially reconfigurable region that can host additional processing components, for example, codecs. Third-party developers can implement and distribute their own components, that is, HDCs, which must comply with an API exposed by the hardware environment. Most likely, the whole application downloaded by the user will be made of a software part containing non-critical parts and a hardware-accelerated portion implemented in the form of an HDC. The FPGA is equipped with two built-in security mechanisms. In particular, the master reconfiguration key (MRK) is a symmetric encryption key used by the on-chip configuration circuitry to decrypt the (partial) bitstreams downloaded to the FPGA. The FPGA device identifier (FID) is a unique code associated with each different physical instance of the device, used to uniquely bind a bitstream to a user device. The FID is non-critical in terms of security and can be possibly accessed from the outside. Notice that both mechanisms reflect existing security-related measures available on current FPGA devices [104, 32].

Two key components are built around the FPGA device on top of the basic architecture. The secure initialization interface is essentially used to initialize the MRK. It is meant to be accessed in a secure environment under the sole control of the TTP and possibly of the FPGA fabric vendor. The SAM is a tamper-resistant device with encryption capabilities,

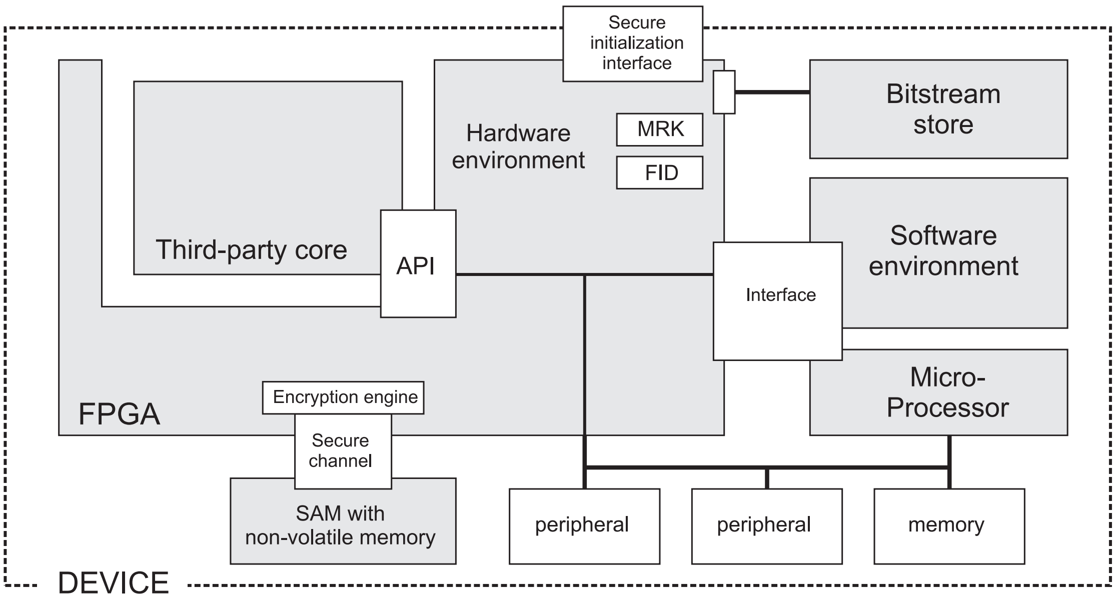


Figure 3.2: Physical architecture of a device compliant with the proposed secure core distribution infrastructure.

supporting symmetric and public-key cryptography and secure non-volatile storage. It is associated with the user identity and can be remotely accessed by the TTP online. In a real scenario, the SAM is likely to coincide with a Subscriber Identification Module (SIM) normally available in handheld devices, smart meters, set-top boxes and all devices that need to have a cryptographic identity. More generally, in trusted computing platforms the SAM will coincide with the TPM, which is equivalent to a smartcard in functionality. Moreover, in the particular case of a mobile device equipped with a SIM card, the TTP can coincide with the Mobile Network Operator (MNO), or can obtain the access to the user SIM from the MNO as a service. The SAM is vital in the envisioned architecture in that it provides a cryptographically secure storage for the ROs associated with each HDC, making the application stateful (unlike the FPGA itself), and linking the device identity with the user identity. Based on the above architecture, the following assumptions can be made concerning the security at the level of the device:

- the architecture surrounding the FPGA is not necessarily secure, in particular it must not necessarily be a trusted computing platform;
- the device can be physically tampered with;
- the SAM, along with the ROs in its NVM, can be removed and replaced with a fake SAM;

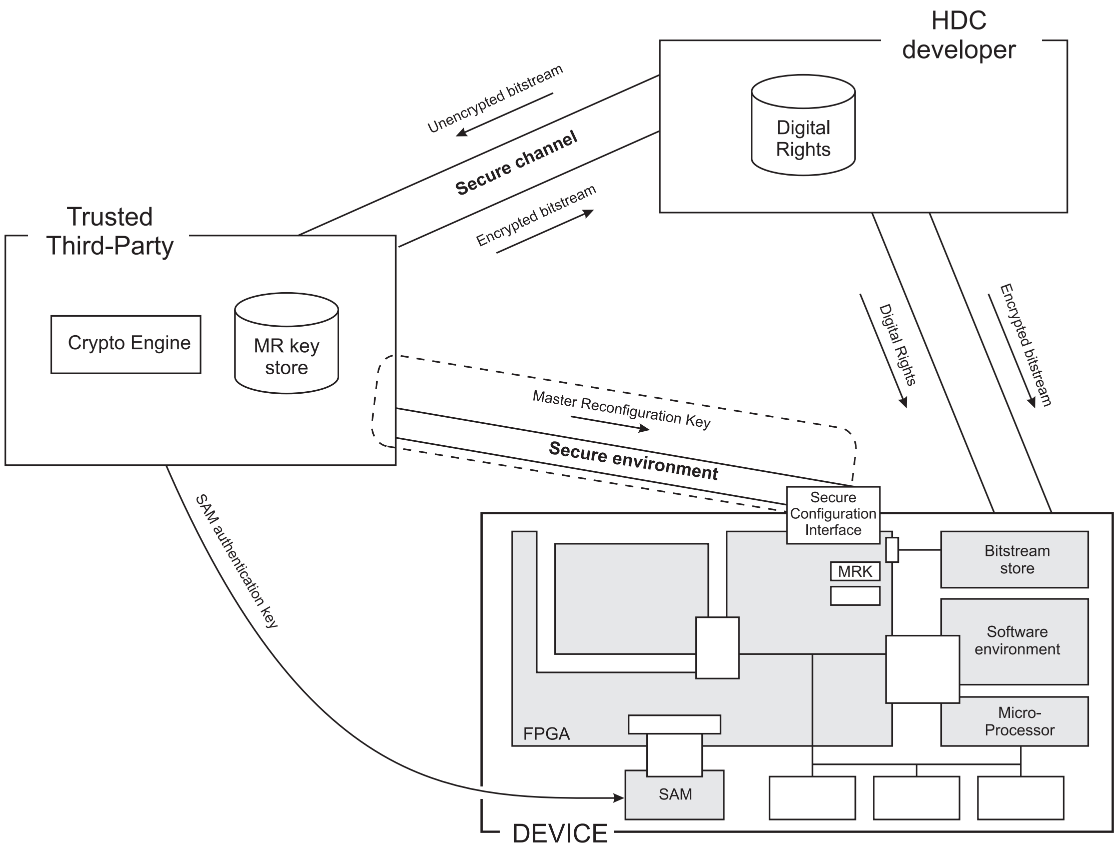


Figure 3.3: Architecture of the secure core distribution infrastructure.

- the secure initialization interface is always accessible for writing the MRK (although the genuine MRK is always initialized in a secure environment).

3.4 Infrastructure Architecture

Figure3.3 describes the proposed infrastructure for HDC distribution along with the interactions between the players.

The MRK initialisation involves the TTP and the End User, relying on the secure initialization interface. The process is write-only, meaning that the MRK can never be readback. Any attempt to access the initialization interface will result in the loss of previous MRK, making the bitstreams specific to this device unusable. Notice that the physical architecture is constrained by existing FPGA security mechanisms. In particular, the MRK key enters the FPGA in a plain form and can be read by probing the device pins during initialization. That implies that the initialization process must always take place in a secure environment, possibly at manufacturing time. Notice that current technologies, such as Xilinx bitstream encryption, allows the non-volatile storing of the decryption key by either storing it in a RAM powered by a small backup battery or by using non-volatile OTP fuses.

The TTP can also interact with the SAM in the device to securely transfer the secret key used by the SAM to authenticate the FPGA requesting access to the ROs. Since the SAM (e.g. the user SIM card) represents a secure perimeter with its own identity and cryptographic capabilities, the interaction between the TTP and the SAM can take place at any time, possibly mediated by the MNO, in a secure way, even when the SAM and the FPGA are placed in an untrusted device.

A bitstream registration/validation/encryption procedure takes place between the TTP and the IPCore Vendor. This interaction takes place online through a standard secure channel, most likely an SSL-secured connection, in order to authenticate the TTP and preserve confidentiality of the HDC bitstream. Having full access to the bitstream, the TTP validates its structure by both checking its compliance with the standard API as well as its trustworthiness, for example, the absence of hardware trojans. Notice that it is very unlikely that the user device can autonomously perform such check directly on their device: On the one hand, that would expose the bitstream outside a secure perimeter and, on the other hand, it would require overly large analysis capabilities to reside on the user device. This is a major reason explaining the role of the TTP.

Once the validated and encrypted HDC bitstream is obtained from the TTP, the IPCore Vendor can distribute it directly to the end user, through possibly unsecure channels. It also creates and issues the ROs associated with the HDC. Figure 3.4 depicts the generic structure of an RO for an HDC. The ROs can define permissions, constraints and obligations enforced on the use the HDC. For instance, a limited number of uses can be granted for the distributed HDC, for example, for a trial installation. The management of the ROs (e.g. the count of the different uses) requires an NVM which, because of the above technological constraints, is hosted outside the FPGA in the SAM. The API available from the static part of the design to the HDC exposes the required interface to authenticate the HDC to the SAM through a challenge-response process before reading/updating the ROs associated with the HDC stored on the SAM. Notice that, as discussed later, the authentication of the FPGA to the SAM is key to guaranteeing a

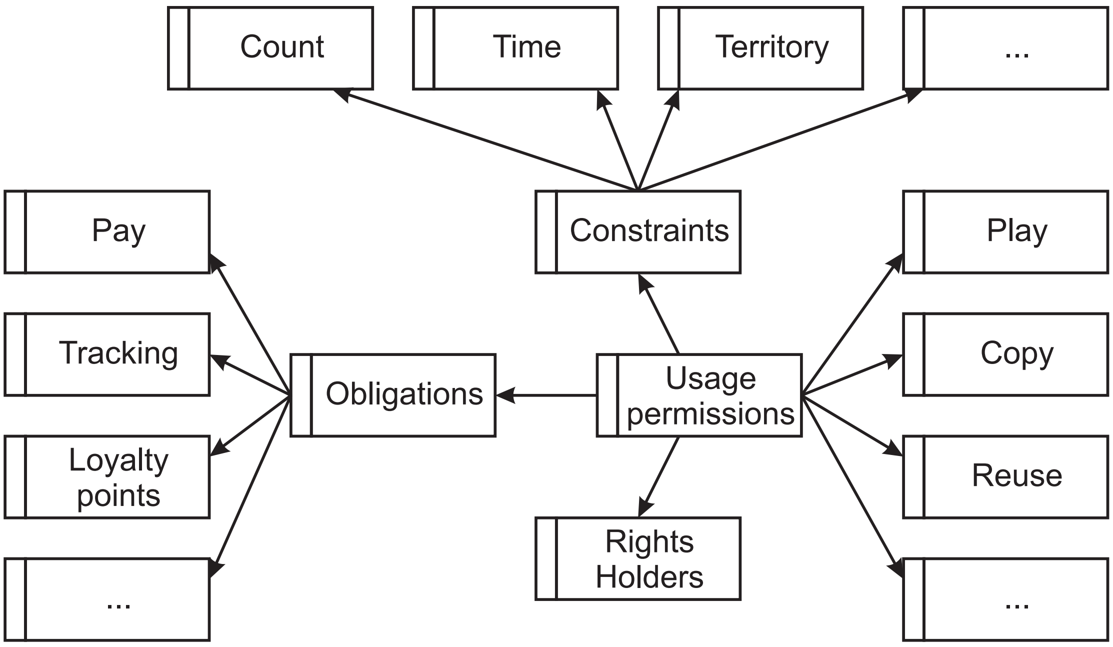


FIGURE 3.4: Structure of a digital RO for HDCs.

trustworthy manipulation of the ROs, which do not reside on the HDC itself because of the lack of NVM in the FPGA.

3.5 Security Evaluation

This section reviews the attack scenarios previously identified and analyzes the impact of the security mechanisms put in place by the proposed infrastructure. For each security threat, more than one scenario is figured out, illustrating how they enable the attack and discussing its potential consequences on the system integrity.

3.5.1 Malicious bitstreams

Genuine host environment This scenario assumes that an IPCore Vendor wants to inject a malicious HDC bitstream into the end user device without the user's collusion. This implies that the software environment on the device is still genuine. The environment natively prevents FPGA configuration outside the secure environment accessed only by the TTP (see Figure 3.2). In particular, no MRK update can take place. As a consequence, only bitstreams previously validated and hence signed by the TTP can be downloaded to the FPGA. Of course, the authorized bitstreams can be deemed safe as long as the TTP is able to successfully detect malicious bitstreams upon validation, for example, hardware trojans. A further level of protection is provided by the API exposed by the static part and the software environment to the partially reconfigurable HDC, which guarantees the isolation of the hosted component from the low-level features of the user device.

Hacked host environment This scenario hypothesizes that the end users is actively involved in the attack in that he/she intentionally allows low-level modifications to the device, for example, replacing system components such as the SAM. No malicious modification, however, is possible within the secure environment accessed by the TTP to initialise the MRK. In case the MRK is stored in an on-chip RAM within the FPGA, by hacking the low-level features of the device, particularly the FPGA configuration logic, the user can inject their own MRK and consequently generate arbitrary bitstreams. However, apart from potentially causing damages to the device, this type of vulnerability by no means allows the generation of false ROs enabling unauthorized access to genuine bitstreams, since these are still enciphered with a legal secret MRK. In fact, hacked bitstreams can interact with the SAM, but they cannot authenticate themselves to the SAM, which will not allow any read/update operation on the stored ROs. Notice that the MRK can never be read back from the FGPA and can only be overwritten in case it is stored on an on-chip RAM powered by a back-up battery. On the other hand, if the MRK is stored in an NVM (e.g. fuses within the FPGA), the MRK cannot even be modified, which would provide improved security (while reducing the flexibility of the distribution infrastructure).

3.5.2 IP theft and reverse engineering

Network sniffing HDC bitstreams exist in an unencrypted form only within the IPCore Vendor's and the TTPs perimeter. Furthermore, they need to travel from the IPCore Vendor to the TTP across an untrusted communication channel. For this type of interaction, however, the channel can be easily secured through standard mechanisms, particularly an SSL connection. After being encrypted, the bitstream can be safely distributed through unprotected channels, for example, through the Internet.

Device hacking Bitstreams are stored in an encrypted form on the device. This makes it impossible to reverse engineer the HDC by reading it from the device configuration store, in addition to ensuring the confidentiality of the keys used for authenticating the FPGA to the SAM and decrypting the ROs. Encrypted bitstreams are decrypted within the FPGA after being downloaded from the configuration store. Consequently, the only possibility of obtaining the unencrypted bitstream at run-time would be to unpackage the FPGA while it is powered up and probing the running configuration. Such a process is extremely unlikely to be feasible in practice. Nevertheless, its cost would be overly large compared to the illegal monetary gain it can possibly enable for the class of applications previously envisioned in this manuscript.

3.5.3 Right Object tampering

RO tampering based on cryptanalysis ROs are securely stored within the SAM, which only grants access to authenticated external components, namely the HDCs running within the FPGA. Furthermore, ROs might be optionally encrypted by means of an RO key (ROK) defined by the IPCore Vendor and hardwired in the HDC. ROs cannot thus be sniffed by physically probing the device. Securing the bitstream against IP theft by encryption also guarantees the protection of the ROKs stored in it.

RO tampering by device hacking Attacks not relying on cryptanalysis might instead consist in hacking the physical device, by replacing the whole NVM containing the ROs, for example, to restore the initial value of a counter. This vulnerability would inherently rely on the fact FGPAs need an external storage for handling non-volatile data. Here the use of a SAM is key, as it provides a secure perimeter where ROs can be stored and securely exchanged. Replacing the SAM has no effect since an authentication process takes place between the FPGA and the SAM. The use of the ROK to encrypt the ROs, possibly chosen by the IPCore Vendor on a per-device basis, provides a further level of protection.

3.5.4 Software hacking

If the device surrounding the FPGA is not a trusted computing platform in itself, the software environment can be easily tampered with. In particular, an attacker can easily reverse engineer the software part accompanying the HDC. In fact, in a real-world scenario, HDCs are likely to implement only a performance-critical kernel of an application (e.g. an intra-frame predictor as a part of a software video codec). By analyzing the interface of the hardware component, the attacker could reproduce in software the functionality implemented by the HDC and hack the high-level application by replacing the FPGA with their own software routine. Obviously, however, even in the case that such hacking can reproduce a functionally equivalent HDC in software, the performance levels would be orders of magnitude lower, providing no practical incentives to the attackers, since the main value carried by a genuine HDC lies in its hardware-supported execution speed.

3.6 Prototypical Implementation and Results

In this section, an implementation of the FPGA-based End User's device is presented, demonstrating the proposed architecture. To this aim, a commercial configurable platform, namely a Xilinx Zynq-7000 XC7Z020 mounted on a Zedboard development kit, is adopted, allowing to prototype both the hardware reconfigurable part and the software environment of the user device (see Figure 3.2). The system includes the Xilinx FPGA, a 512 MB DDR3, 256 MB Quad-SPI Flash, SDCard slot, Ethernet, USB OTG, HDMI/VGA and audio lines. The FPGA is a System on Programmable Chip; indeed, it has a fixed processing system that can operate independently of the FPGA Programmable Logic (PL). The core of the processing system is an ARM dual-core Cortex A9 32/32 KB I/D caches, 512 KB L2 cache. Some peripherals are available to the processing system, including the Device Configuration (DevC).

The DevC allows the processor to download full/partial bitstreams to the FPGA PL. It is also equipped with a built-in AES decryption engine, coupled to the HMAC engine, that can be activated to allow the configuration of encrypted bitstreams. The HMAC provides private key authentication using the SHA-256 hash function. The AES encryption key (i.e. the MRK) can be stored in the Battery Backed RAM (BBRAM) or in eFUSE array. The BBRAM is a reprogrammable/zeroisable memory that can be considered an NVM if a battery power supply is provided, because the battery guarantees the key storing when the device is powered off. The eFUSE is an OTP NVM, called eFUSE array. Both the registers provide secure storage because they are inaccessible to an attacker. In fact, all the memory components (on chip memory, L1 and L2 cache, AXI block RAM, PL configuration memory, BBRAM and eFUSE) reside within the security perimeter of the FPGA [98]. Both BBRAM and eFUSE can be programmed with the AES key by using the JTAG interface with Xilinx iMPACT tool. In addition, eFUSE contains some other configuration bits, such as the disable read bit, that inhibits the AES key readback. Once these bits, including the key, are written in the eFUSE, they cannot be changed and, if the readback is disabled, they can never be read.

3.6.1 Software environment: Zedroid

In order to prototype the software stack of the user device, the FPGA was equipped with a software environment tailored for the Zedboard, obtained by porting the Google's Android operating system for the Zynq platform, called Zedroid. The porting involved adapting the whole Android software stack and services on the Zynq PS core, supporting possible hardware extensions [14]. For instance, in this specific case, the Linux device tree had been patched to make the DevC and the custom accelerator visible to Android, and the kernel to support the DevC driver that manages the DPR process. As for the PL hardware, the Zedroid porting provides a basic HDMI core interface using the on-board ADV7511 chip and a blank processing accelerator, both defined with an AXI interface. By using the USB OTG protocol, the system accesses a SIM card reader to the system. The SIM includes a secure NVM which contains the ROs, providing information about the HDC payments/uses, because the FPGA is not equipped with an internal secure NVM. Figure 3.5 shows a schematic view of the described architecture. Concerning the hardware components, the FPGA hosts the HDMI and the AES decryption engine; concerning the software, the Android stack includes the kernel objects (KOs) and the user application that will be discussed in the next Subsections.

3.6.2 HDC lifecycle managing framework

To manage the installation of the accelerators via partial reconfiguration mechanisms, a framework that exposes some APIs has been developed. The APIs guarantee the dynamic configuration of a new hardware unit, by means of the DevC, within the Java code. As highlighted by Figure 3.5, the DevC driver structure is made of two parts. The low-level part, integrated in the Linux Kernel and User space, avoids inconsistent state of

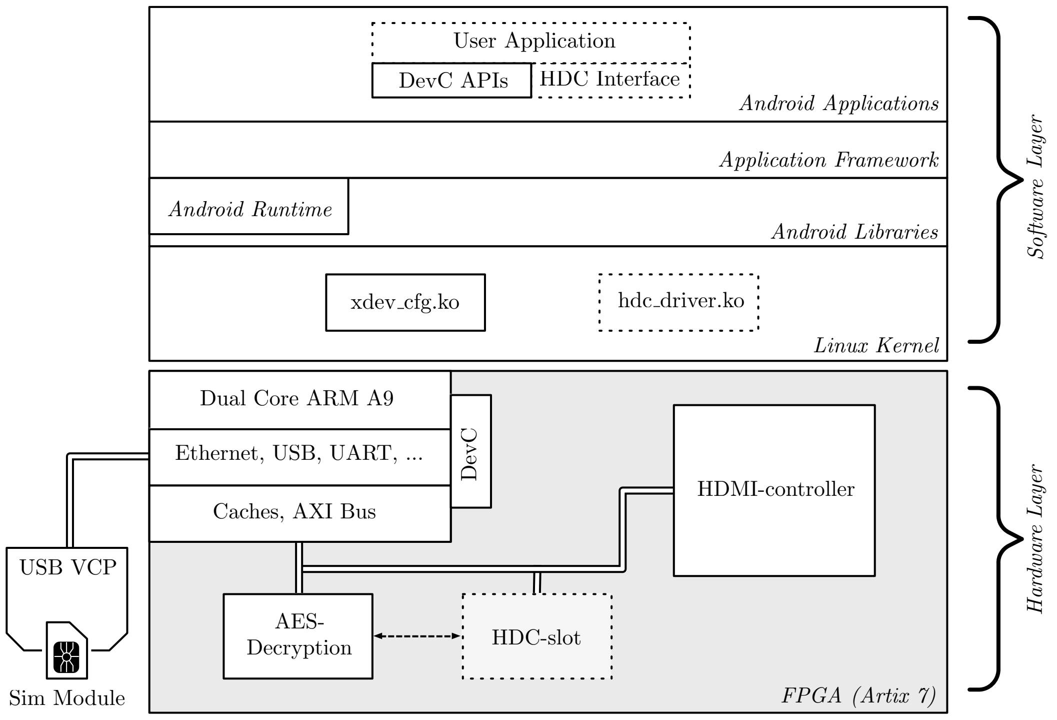


Figure 3.5: Structure of the Zedroid architecture, equipped with the software and hardware components in order to support pay-per-use scheme.

the system. For instance, it provides synchronization mechanisms, by exploiting a kernel mutex, that resolves contentions when multiple partial reconfigurations are requested. This mechanism is replicated for each accelerator slot available on the PL, but for the experiments only a single accelerator has been considered. The high-level part of the driver is implemented by means of JNI functions: they allow an application running on the Dalvik Virtual Machine to call user space functions directly. By using these APIs, an Android Activity, or Service, requests for a partial reconfiguration. If the accelerator is not configured, the operation succeeds, else an error state is returned. Since at startup no HDC is installed, any attempts of accessing a blank device are forbidden by the low-level driver. Only after a partial reconfiguration operation it will be possible to access the new peripheral. When an Activity is to be destroyed, or paused, the accelerator is set free to grant other Activities access to the configuration of the accelerators. The behavior of the Activity on those events is a developer's concern, because the use of the accelerator is tightly coupled with the Activity functions. After each installation, the accelerator is activated by checking a proper RO. This means that the Android App needs to query the SIM to receive an enciphered token used to activate the accelerator. The implementation of the token checking is an IPCore Vendor's concern. This solution involves both the IPCore Vendor and the TTP in the bitstream billing and access process.

3.6.3 Integrating dynamic hardware components

As for the HDC interaction with the user space software, the Zedroid project provides useful tools to easily integrate new hardware components. First of all, into the Device Tree of the Zedroid source files, a generic peripheral is declared so that, at boot time, the Linux Kernel is ready to load drivers for the dynamic devices. This peripheral is referred as a dummy or blank device. The configuration required for a Device Tree entry are the physical address, a unique ID, the interruption manager and so on. These values, for dummy device, are fixed and generic in order to support a high range of peripherals. To deploy new peripherals in the Zedroid environment (we can refer it as hdc_device), the software designer needs to define a KO which manages the interaction between the user, that is, Android applications, and the hardware component. To this aim, the Zedroid project provides a driver template which contains the functions needed to dynamically mount the hardware, by loading it in the kernel. Since the obtained driver is a KO, the system needs to run insmod to dynamically mount new drivers onto Linux Kernel space. This command installs a loadable module (i.e. driver) in the running kernel (Figure 3.5). If the procedure succeeds, the peripheral will look like a file under/dev system folder (e.g. /dev/hdc_device). All the information about the peripheral is retrieved from the Device Tree dummy entry, because the drive template file declares itself as compatible with the dummy device. To unmount the driver, in order to change it with another one, it is necessary to call rmmod. The template main functions are the file operations (read, write and seek), which implement the access logic to the peripheral. By using Java File class, it is possible to access this peripheral from the Android user space, instantiating a File object pointing to the right path (/dev/hdc_device). In the Zedroid source a Java Class, named ZedPeriph is available. This class, implemented with Singleton design pattern, declares some useful functions (such as open, close, seek, read and write) used to easily access the peripheral. By inheriting it, a developer can easily implement the main driver functionality, that is, the control, data transfer and status logic.

3.6.4 Configuration time overhead

In order to derive a quantitative analysis of the overhead introduced by the support for HDC distribution, some experiments were ran. In particular, the analysis regards the overhead required by the partial reconfiguration task using partial encrypted bitstreams and the overhead required for the SIM communication. As for partial reconfiguration, the differences between the plain and ciphered bitstreams are highlighted. To this aim, five partial block rectangles were designed, varying the sizes of the occupied resource, starting from the hardware project described above. The block sizes are as follows: $1600 \times \{1; 1.5; 2; 3; 6\}$ LUTs. For each partial block definition, two different hardware accelerators with an AXI stream interface were synthesized and the obtained partial bitstreams converted from the .bit to .bin format by using the Promgen software tool. Also, in order to measure the time to complete each partial reconfiguration process, the DevC driver was instrumented to count how many microseconds occur between the start and the end of the DMA transfer process. Each experiment was run nine times to average out the interferences caused by the Android OS and underlying Linux Kernel. Figure 3.6 shows the partial reconfiguration delay curves varying on the size of the bin file, referring to the plain and ciphered bin files, obtained by linear interpolation of the experiments. The values of the experiments related to the two different accelerator implementations were merged, because it is possible to observe that the configuration overhead depends only on the file size. The configuration delay trend is linear with respect to the size of the .bin in both cases, but the slope of the ciphered case is about four times larger than the plain one. In fact, the PCAP module in the DevC works with a frequency clock scaled down by a factor four, directly impacting the effective maximum throughput.

3.6.5 Case-study and experimental results

To demonstrate the proposed solution, a further experimental activity was conducted where the main interactions of the proposed distribution infrastructure were reproduced. From this additional test activity, the related overhead was quantified. Assume that a media player application, downloaded from the app store, includes a hardware accelerator, to be dynamically configured onto the system as an HDC, available at an extra fee. This accelerator implements the H.264 intra-frame prediction in order to enhance

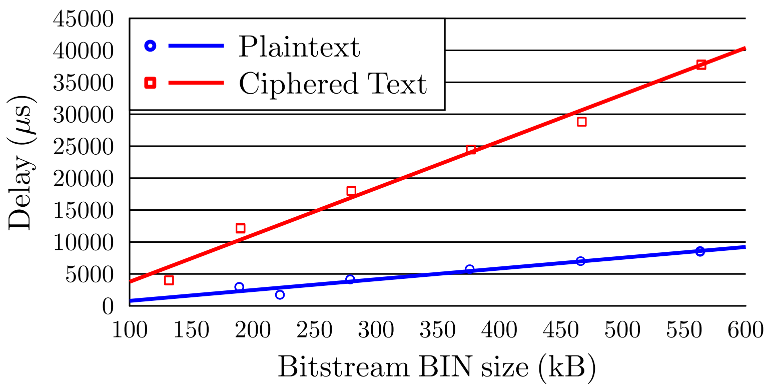


Figure 3.6: Delay curves related to the partial reconfiguration of plain and encrypted bitstreams, varying with the bitstreams file size.

the frame-per-second rate. By accessing the billing system, provided by the TTP as a normal online store, the End User is able to complete the payment transaction and get the access to the accelerator, provided by the application. This right is issued by the TTP as an RO and has to be written in the SIM. The accelerator is downloaded by the application as a partial enciphered bitstream by means of a URI. Once the application is configured by the Android Activity, it interacts with the SIM to see if it has the rights to access the accelerator. The SIM returns a ciphered token used by the Android Activity to activate the accelerator and to start the video playing. The application implements all the H.264 decompression process in software, except the intra-frame prediction, implemented by the installed HDC. This designed prototype was used to measure the time overhead required by RO managing in such a scenario. The SIM module communicates through a USB VCP at a baud rate of 9600 bps. The time requested to save a RO and to query the SIM for the RO were measured: the saving process requires on average 241 ms, the query about 182 ms. As for the hardware accelerator, the IPCore contains an H.264 infra-frame prediction and a 256-bit tiny AES core – used to process the RO on chip – requiring 6231 LUTs, 6752 Slice Registers and 98 Block RAM, that is, around 10% of the available resource. The overall overhead, measured from the DevC DMA starting time to the HDC activation, is 867 ms on average.

3.7 Considerations on the Approach Improvement

In addition to the definition of the infrastructure and device architecture, a major lesson learned from this experience lies in the analysis of the security-related features and

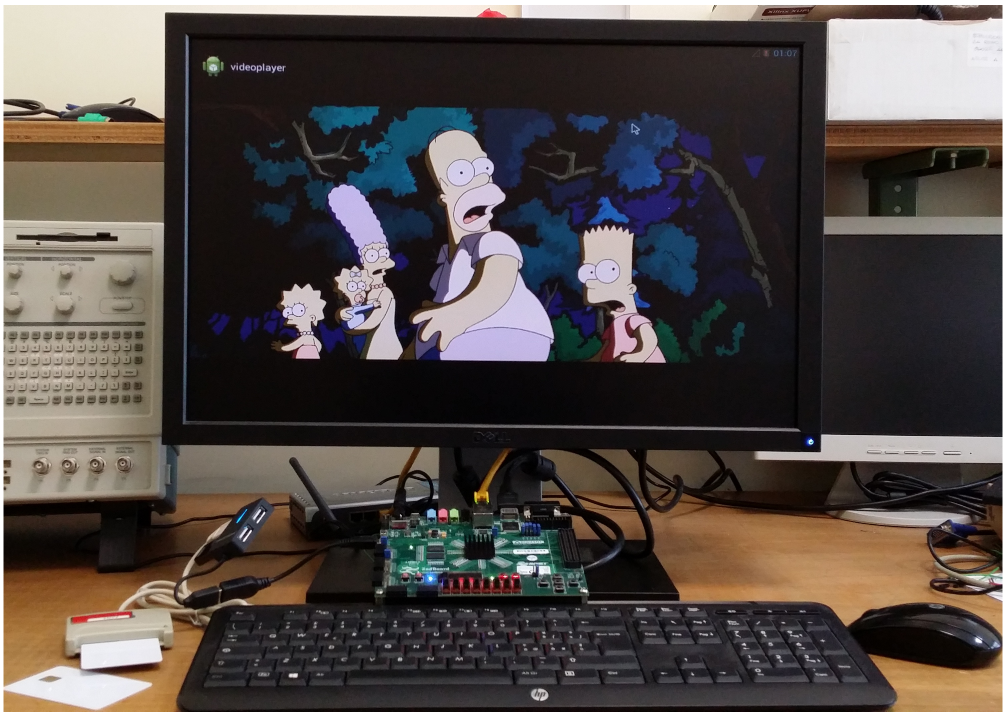


FIGURE 3.7: In the photo the Zedboard is running the Zedroid software stack on which it is downloaded an Android video player app with an hardware accelerator for the H.264 infra-frame prediction.

limitations of current FPGA devices. This Chapter is concluded by drawing a few proposals for the evolution of FPGA features that may enable the full realization of the secure HDC distribution concept:

- **Separate encryption keys**: current FPGAs rigidly use a single key for encrypting both static and partial bitstreams. This limitation inherently requires that the TTP, that is, the manager of the security architecture (embracing the static reconfigurable part of the user device) and the HDC developer (involving the partially reconfigurable region) rely on the same secret key. However, since there may be many independent developers, and each developer should not access the IP of other developers, the only solution was to let only the TTP know the encryption key and encipher the bitstreams in place of all the developers. By supporting separate encryption mechanisms for static and partial bitstreams, this fundamental limitation would be overcome.

- **On-chip FPGA authentication**: current FPGAs do not provide a secure perimeter with a cryptographic identity (possibly, based on public-key cryptography). So, no authentication and secure remote communication is possible between the FPGA and a remote party, for example, the TTP. On the other hand, this is inherently possible with the SAM. However, being physically separated, the FPGA and the SAM must authenticate with each other for preserving the security of the interactions, for example, managing and transferring the ROs. This complication would be completely overcome if the FPGA would be provided with an on-chip authentication mechanism, which would allow the FPGA to directly establish a secure channel with the TTP. This would greatly simplify the interactions. In particular, the MRK initialisation process could take place remotely at any time, without requiring a secure environment.

- **On-chip NVM**: having NVM on the FPGA would remove the need for an external SAM device. The internal memory would be only used to make the FPGA stateful and store some information needed to track the state of the ROs. The bulk information, that is, the ROs themselves, may still reside outside the FPGA, possibly in an encrypted form on an external flash device. The state stored within the on-chip NVM would be essentially only used to prevent replay attacks, possibly re-encrypting ROs at each use so that the user cannot simply restore the initial value of the external NVM to gain illegal uses of the HDC. This solution would only require a limited amount of NVM to be available on the FPGA.

- **Reusing built-in encryption cores**: the prototypical implementation is equipped with an AES core on the static part of the FPGA. This is only required by DRM-related operations. On the other hand, the FPGA is already provided with an AES hardened core, required to decrypt the (partial) bitstreams, and only used at configuration time. Making such core available to the DRM-related operations would of course save a significant amount of hardware resources that could be exploited for the user design.

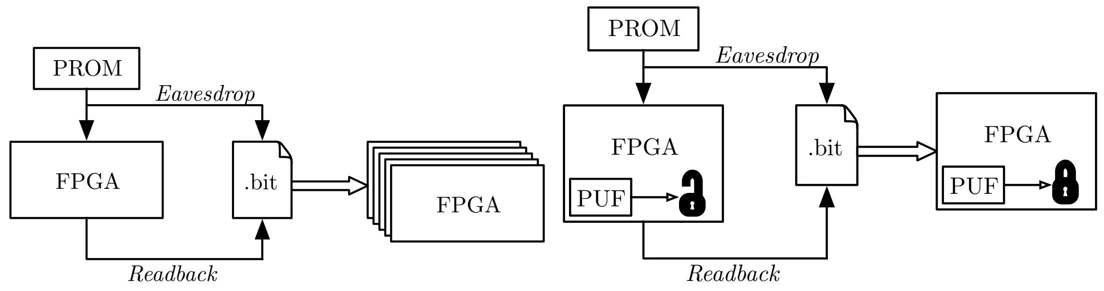
Chapter 4

Enable Security Through Physically Unclonable Functions

The previous Chapter has demonstrated the feasibility of protecting IP trough cryptography, also enabling powerful pay-per-use mechanism through the DRM architecture. But the proposed solutions relies on the availability of built-in decriptor, such that it is possible (partially) configure the FPGA by means of enciphered bitstreams. Old-family or low-end FPGAs were designed without any security mechanism, hence they are still vulnerable to the attacks listed in Section 2.1.

In this scenario, PUFs, previously introduced in Subsection 2.2.2, are an emergent solution to prevent at least bitstream theft and device cloning. All the other attacks are still effective, since the involved bitstreams are in plain and the FPGA has no inherently countermeasures. Through a binding mechanism, indeed, a design in the bitstream can be forced to work only on a particular device. Normally, the bitstream produced for a particular device family is able to be configured on any device which belongs to such a family. In this case, any attack that is able to retrieve the bitstream enables to clone a targeted device, as illustrated in Figure 4.1A. But exploiting a PUF, even if attackers obtain the the bitstream, they will be not able to use it on other devices, since, exploiting the unclonability (mathematical 2.2a and physical 2.2b) and uniqueness (2.3) properties, there are not FPGAs with the same PUF.

Recently, a bitstream binding technique has been proposed by Zhang et al. in [123, 122, 125]. In particular, authors proposed the first non-encryption IP binding method,



(A) Eavesdrop and readback attacks succeed on bitstream without any protection.

(B) Eavesdrop and readback attacks cannot clone devices due to bitstream biding through a PUF.

Figure 4.1: Differences between bitstreams with and without binding to a specific device's PUF.

which combines the unclonable PUF responses for a particular device with a finite state machine (FSM). Interestingly, the adoption of sequential circuits in FPGA designs/IP cores enables to restrict the bitstream to running only onto authorized FPGA chips. The FSM is designed such that both states and transitions depend on the PUF responses. The FSM checks its current state, and if the reached state is a *lock state*, it means that the PUF response is incorrect and that the device is not authorized to run the IP, thus the IP core is disabled. Its implementation is based on the concept of augmented FSM, which is a regular FSM with the addition of an exponential number of states and transitions, such that extracting its State Transition Graph representation is computationally intractable, making reverse engineering unfeasible. Moreover, Zhang et al. have shown a pay-per-device licensing mechanism exploitable by IP vendors which release licenses specifically for an FPGA device. Similarly to the scenario drawn in Section 3.2, the licensing mechanism involves the FPGA vendor, the authors devised a prototyping architecture the FSM-based lock mechanism proposed in the binding scheme. As anticipated in the previous Chapter, also Gören et al. in [45] have introduced a similar approach for a low-end device family, i.e. Spartan-6, which enables the secure configuration of partial bitstreams through a binding mechanism based on a FSM.

Therefore, to enable such a mechanism on old and low-cost FPGA families, the design of trustworthy PUFs is crucial. Not all the available architectures are suitable to implement on the FPGA technology. Indeed, some PUFs, such as the arbiter PUF (Figure 2.5), require symmetric routing. This condition is hard to meet in FPGA design since the routing resources are fixed in foundry and it is not possible to control the path to have a perfect symmetric conditions on involved delays.

To this aim, during the doctoral research activity, two valid candidates have been studied. One is represented by the Anderson PUF, which is logically simple and requires few resources compared to other PUFs, but it is specifically designed for Xilinx Devices. Indeed, the two above cited research works have adopted the Anderson PUF, respectively for the Xilinx Zynq-7000 and Xilinx Spartan-6 families. The other one is the Ring Oscillator (RO) PUF, which is the most investigated PUF since the RO primitive is easily exploitable on all the silicon technology, hence the RO PUF can be even considered as an universal PUF.

4.1 The Anderson PUF

The Anderson PUF can be mainly characterized as reliable, scalable, and easily to size. As indeed, the Anderson PUF is composed by some primitive elements, defined as Anderson cells, which output only 1-bit PUF response. The structure is composed of two shift-registers, two 2-to-1 multiplexers and one D flip-flop. Each shift-register is coupled with a multiplexer, and the output is synchronous with the clock signal and generates an alternating sequence of logic-0 and logic-1. However, their waveforms are complementary to each other. When one outputs a logic-1, the other shift-register outputs a logic-0, and vice-versa. Moreover, each multiplexer address signal is driven by the output of a shift-register, and the *0* data input of multiplexers is stuck to logic-0. Figure 4.2 reports the structure of the Anderson PUF cell, consisting of the elements drawn with solid lines. Multiplexer *B* has its *1* data input tied to logic-1 (depicted with the grey color in Figure 4.2), and its output *N1* is connected to the *1* data input of the multiplexer *A*. The output of the multiplexer *A* (the *N2* signal) drives the preset input port of the D flip-flop, which is initialized to logic-0. Since the preset and clear ports are asynchronous (independent from the system clock), the value held by the flip-flop may be changed to logic-1 from logic-0 by a positive glitch that appears on the preset port. Furthermore, the flip-flop is configured as one-catcher. Therefore, when the flip-flop memorizes logic-1 it will remain in this state indefinitely.

From a logical perspective, since shift-register outputs are complementary to each other, the signal *N2* is always logic-0. However, although the two pairs of shift-register/multiplexer should be physically identical, their delays are different due to intrinsic manufacturing imperfections. Indeed, we can list two cases:

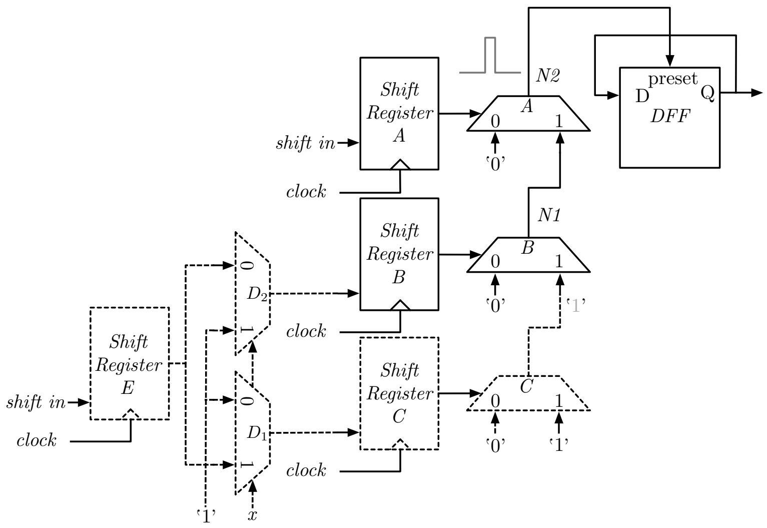
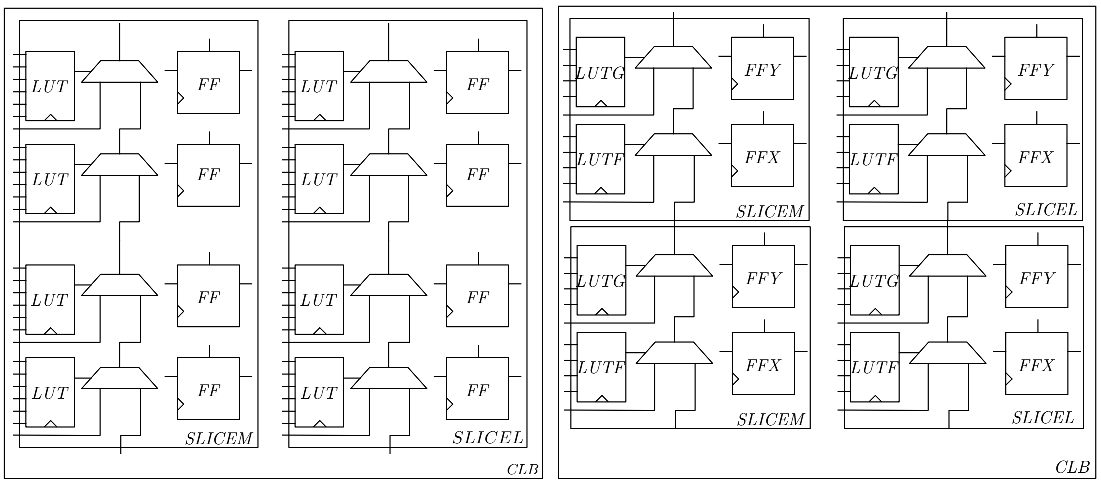


FIGURE 4.2: Logic schematics of the Anderson PUF cell. The solid lines draw the original Anderson PUF [8], and the dotted lines indicate the architecture with a 1-bit challenge response, proposed by Hori et al. in [54]

1. if the delay of the A pair is shorter than the B pair, when shift-register A shifts from logic-0 to logic-1, multiplexer A selects the 1 data input, but signal *N1* has not yet transitioned from logic-1 to logic-0. Therefore, signal *N2* is determined by the value of signal *N1*, which is logic-1 until it transitions to logic-0. During this time interval, a glitch will appear on *N2*;

2. if the delay of the A pair is greater than the B pair, when shift-register B shifts from logic-0 to logic-1, the A multiplexer has not yet selected the 0 data input. Again, signal *N2* is determined by the value of *N1*, which is logic-1, and a glitch will appear on *N2* until the A pair transitions from logic-1 to logic-0.

The glitch shape, in particular its width, is determined by the difference of involved delays. Anderson claimed that if the glitch is too short, the routing network might be able to filter it before it reaches the preset input of the flip-flop [8]. But to be effective, the glitch width has also to be greater than the amount of time required to effectively preset the flip-flop (time constraint). Hence, a glitch appearing on *N2*, and consequently on the preset input of the flip-flop, does not directly imply that the PUF cell response is logic-1. Therefore, a glitch tuning process is required during the PUF design.



(A) Virtex-5 architecture. (B) Spartan-3E architecture.

Figure 4.3: High level view of Xilinx FPGA Configurable Logic Blocks for Virtex-5 and Spartan-3E architecture.

4.1.1 Anderson PUF implemented on Virtex-5

Virtex-5 is built on a 65 nm process, and similarly to other Xilinx FPGAs, it is based on Configurable Logic Blocks (CLBs) arranged in a two-dimensional array (Figure 4.3A). A CLB comprises a pair of slices, each containing four 6-input LUTs, four flip-flops and other logic elements. Slices can be of two types: (i) slice L, which provides logic, arithmetic, and ROM functions; (ii) slice M, which includes the slice L functions, and also implements memory functions, such as distributed RAM and shift-registers. Each CLB can contain either two slices L or one slice M and one slice L. Both slices are equipped with a carry chain structure which can be used to propagates carry signals only through column-aligned slices.

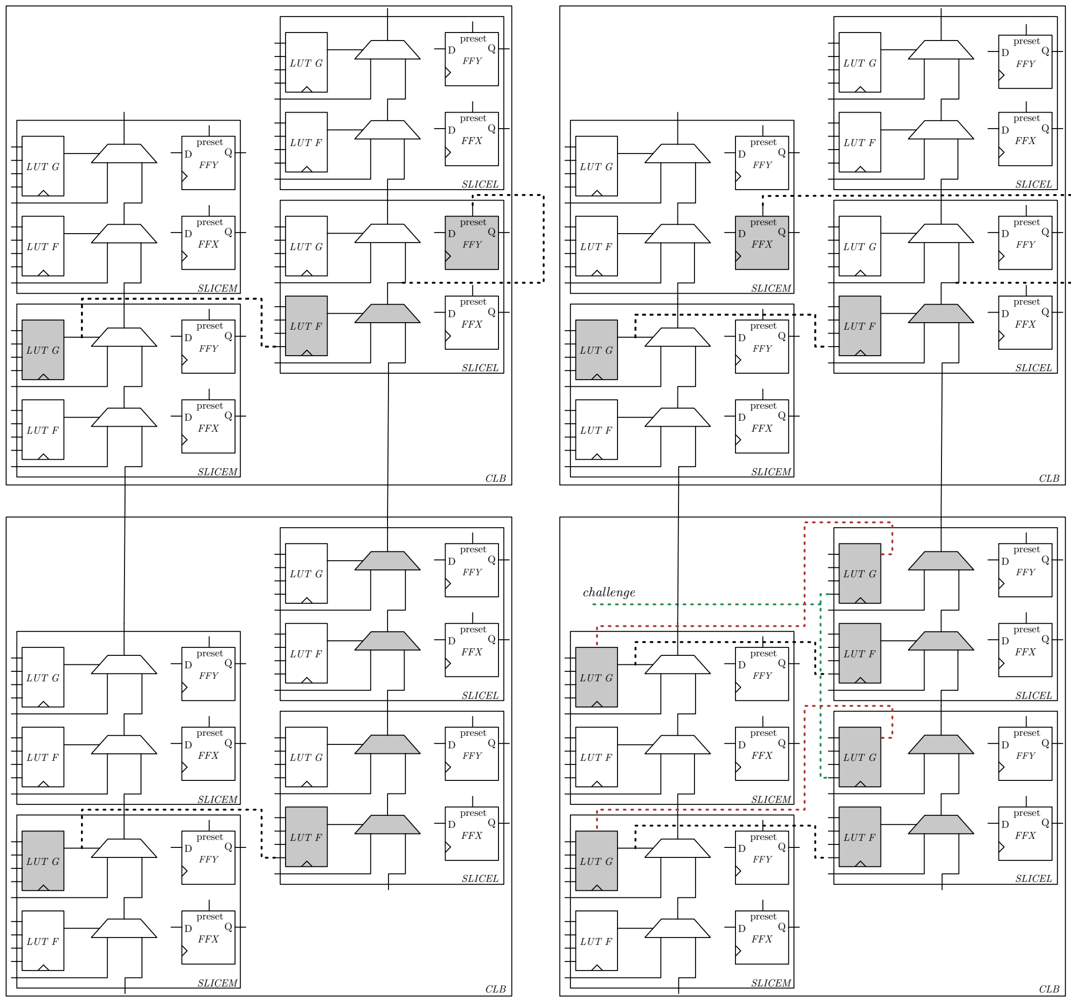
Shift operations of the Anderson PUF are implemented using two slices M, in particular two LUTs configured as 16-bit shift-registers. The two multiplexers are interconnected using the carry chain, and the flip-flop can be located in the same slice in which the other elements are allocated. Therefore, the Anderson PUF cell could be theoretically implemented using only one slice, but glitch modulation has to be considered. To this aim, Anderson increased the glitch pulse width by varying the distance between the two multiplexers. The best tuning was shown to be composed of 5 intermediate carry chain multiplexers. Indeed, Anderson stated that this configuration produced the best results in terms of responses quality [8].

4.1.1.1 Anderson PUF implemented on Spartan-3E

Spartan-3E architecture (Figure 4.3B) is different from the Virtex-5 structure. Indeed, each CLB is formed of four slices grouped in pairs, and each pair is organized as a column with an independent carry chain. Each slice is mainly composed of two 4-input LUTs, namely F-LUT and G-LUT, and two flip-flops, namely FFX and FFY. Right pairs slices belong to the slice L type, while left pairs belong to the slice M type.

Implementing the Anderson PUF on Spartan-3E involves the testing of different PUF cell architectures, since it is possible to act on several design parameters, which represent the degrees of freedom of the PUF cell design process. Indeed, the flip-flop can be either implemented using the FFX or the FFY storage element, and it can also be placed into the same slice used for the others PUF cell elements or into a different one. Similarly, it is possible to vary the multiplexers locations by picking either the slices M or the slices L, but having them on the same column, such that they can use the same carry chain. By varying the multiplexers and the flip-flop locations, it is possible to modulate the glitch pulse width. Furthermore, shift-registers can be either implemented by F-LUT or G-LUT of slices M. However, from experimental measurements, this parameter resulted to be a non-influencing factor as long as the symmetry constraint is met, meant both shift-registers should be mapped in the homologous LUTs. Additionally, it is possible to vary both the number of shifting bits and the initialization values of the two shift-registers, originally configured as 16-bit shifting *0x5555* and *0xAAAA* values. Indeed, Huang et al. claimed that these values are not suitable for generating an unbiased PUF response, because there is a good chance that a positive glitch will appear independently by which shift-register/multiplexer pair is faster [54]. Hence, they proposed *0x8888* and *0x4444* values, and they showed that using this configuration better results could be obtained. Nevertheless, these values have been tested with all the implementations, but the obtained PUF response has been always equal to 0. Hence, in the further discussions these configurations will be not considered.

After testing several different PUF cell implementations, it is possible to identify the best PUF cell architecture (Figure 4.4A), by means of quality parameters (see Subsection 4.1.3). The selected architecture requires 5 slices and 2 CLBs, shift-registers are implemented using G-LUTs and their output is fed back to their input in order to allow the same output sequence to continue beyond the initial 16 cycles. Moreover,



(A) Basic Anderson PUF cell. (B) Enhanced Anderson PUF cell.

FIGURE 4.4: Implementations of the Anderson PUF cell.

shift-registers A and B are configured to shift the 16-bit values of *0x5555* and *0xAAAA*, respectively. Multiplexers are implemented using the slice L carry chain, resulting in a cascade of 5 multiplexers. The D flip-flop is located into the same slice of multiplexer A, and it uses the FFY storage element.

4.1.2 Enhanced Anderson PUF

The Anderson PUF cannot work in a challenge-response paradigm, but only as unique signature generator, since input (challenge) signals are not involved. To address this issue, authors in [54] proposed the PUF cell structure shown in Figure 4.2, which is formed by the original PUF cell elements (depicted with solid lines) and additional ones (depicted with dotted lines). Indeed, one extra multiplexer C is located between the A

and the B one, and two additional multiplexers (D_1 and D_2), accordingly with the value of the challenge x, drive the output of the extra shift-register E into the input of the shift-registers B and the C. Shift-register E and A are configured to output an alternating and complementary sequence of logic-0s and logic-1s. If $x = 1$, the multiplexer D_1 feeds the input of the shift-register C with the output of the shift-register E, while the multiplexer D_2 feeds the input of shift-register B with a constant logic-1 value. Therefore, the multiplexer C outputs depend on the shift-register E output, and the multiplexer B forwards its output. Conversely, if $x = 0$, the multiplexer D_1 feeds the input of the shift-register C with a constant logic-1 value, while the multiplexer D_2 feeds the input of the shift-register B with the output of the shift-register E. In this case, the multiplexer C outputs a constant logic-1, and the multiplexer B can either output a logic-0 or a logic-1 accordingly to the output of the shift-register E. Therefore, the purpose of the challenge signal is to pick two multiplexers in order to compare their delays. The allocation of multiple enhanced Anderson cells does not request to replicate each time the shift-register E, because its role is just to provide the right sequence to the shift-register B or C accordingly with the challenge signal.

Following this procedure, signal x for each PUF is one challenge bit, and considering N PUF cells, the PUF is able to generate 2^N responses. However, since the Anderson PUF is a concatenation of elementary cells which outputs are not dependent on other challenge inputs, the output of the PUF is easy to predict. Indeed, a simple attack is trivial to be successfully accomplished, considering two responses from challenges which all bits are fixed to logic-0s and logic-1s. Thus, compounding bits from the two previous responses, which rely on given challenge bits, it is easy to exactly emulate all the 2^N possible PUF outputs. Indeed, it is necessary to collect two responses from the PUF with only two challenges inputs (e.g. bit-string of zeros and bit-string of 1), then, according to the challenge to predict, the proper responses bits have to be picked. To address this issue, more challenge bits have to be integrated into the PUF cell design by following the procedure described above, or using a hash-function which produces responses hard to predict, e.g. $r' = hash\,(c \oplus PUF\,(c))$.

4.1.2.1 Enhanced Anderson PUF implemented on Spartan-3E

The degrees of freedom of the Enhanced Anderson PUF cell are the same of the Anderson PUF cell. However, since more logical elements are required, there are fewer possible implementations for the matching of the Enhanced PUF cell into 2 CLBs.

Among all possible implementations, which are in number less than the porting of the previous architecture, the best PUF cell architecture have been identified (Figure 4.4B). It requires 7 slices and 2 CLBs. The flip-flop is positioned into the slice M slice above the shift-register A one, and it is implemented using the FFX storage element. The external shift-register E is configured to output *0xAAAA*, and the A one with *0x5555* value. Both shift-registers have their output connected to their input in order to allow the same output sequence to continue beyond the initial 16 cycles. Moreover, the output of the shift-register E drives the input of the two additional multiplexers D_1 and D_2, which, according to the challenge signal value, generate the configurations for the shift-registers B and C. Therefore, one of the two shift-registers outputs *0xAAAA*, and the other all logic-1s. In particular, multiplexers D_1 and D_2 are implemented using *G-LUT*s of the same slices where multiplexers B and C are implemented. As for the basic Anderson PUF, the optimal carry chain length turned out to be composed of 5 multiplexers.

4.1.3 Experimental validation

This Subsection illustrates the experimental results obtained from the PUF architectures described above in Subsections 4.1.1.1 and 4.1.2.1. In particular, 15 Xilinx Spartan-3E FPGAs (XC3S1200E) on Digilent Nexys 2 boards involved in each experimental campaign. Since the XC3S1200E FPGA has 60 CLB rows for each CLB column and each Anderson PUF cell (for both the architectures) requires 2 CLB rows, it is possible to instantiate 30 PUF cells on a single CLB column, thus achieving a 30-bit PUF response per CLB column. Each experiment involved a 60-bit PUF response, hence 2 entire CLB columns were required. Furthermore, the FPGA was split in 11 non-overlapping vertical regions, resulting in 11 different 60-bit PUF implementations per FPGA, thus $11 \times 15 = 165$ different PUFs. Their responses were sampled at the FPGA start-up using Xilinx ChipScope Pro tool. However, the Enhanced Anderson PUF involves a different procedure since it requires challenge inputs. Therefore, two challenges were adopted, one composed of all logic-0s and one composed of all logic-1s, and a different ChipScope instance in order to set the challenge bits and to clear the value held by the flip-flops. The following procedure has been applied for each challenge:

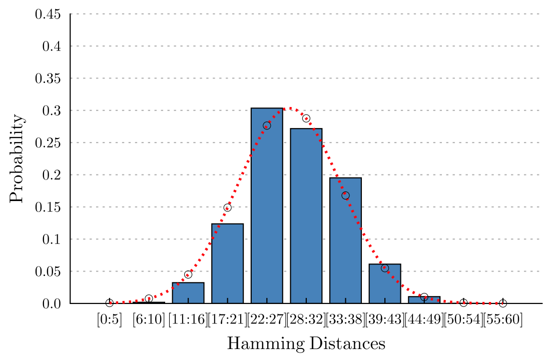
- the signal *clear* is set to logic-1 in order to reset the values held by the flip-flops;
- the challenge is applied;
- the signal *clear* is set to logic-0, hence the flip-flops are free to latch any positive glitch appearing on their preset input ports;
- the response is sampled.

In order to evaluate the quality of the PUF architectures, the gathered responses were analyzed with some quality metrics, which goal is to evaluate how the PUFs are close to some ideal properties given in Subsection 2.2.2.1. Further results are briefly summarized in Table I, along with some other delay-based PUFs available in the literature.

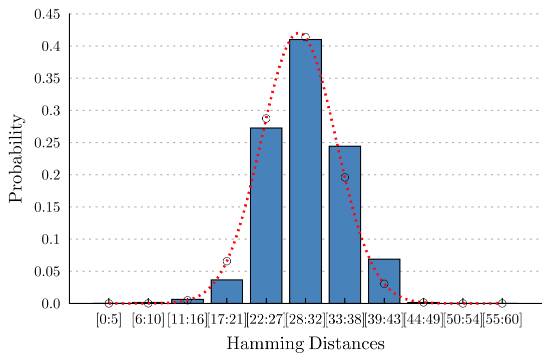
4.1.3.1 Global uniqueness

In order to adopt PUFs in security applications, their responses have to be unique among a population of devices. To this aim, the global uniqueness metric can be adopted in order to measure the variation of PUF responses to the same set of challenges, but on different devices. The global uniqueness can be obtained as the average fractional Hamming Distances (fHDs) between all the PUFs responses pairs extracted from the combination of different devices, FPGA regions, and challenges (if any). The HD is a function on two binary strings of equal length which returns the number of homologous bits that differ, or, in other words, the minimum amount of substitutions needed to change one string into the other. The fHD normalizes the HD between 0 and 1, dividing the HD by the string length. Moreover, let R be the number of PUFs responses resulting from the product between the amount of different devices (15) and the PUF instances (11). Let r_i and r_j $(i \neq j)$ be two of the R N-bit responses, the percentage measure for the global uniqueness can be defined as:

$$Uniqueness = \frac{2}{R(R-1)} \sum_{i=1}^{R-1} \sum_{j=i+1}^{R} \frac{HD(r_i, r_j)}{N} \times 100\%. \tag{4.1}$$



(A) Basic Anderson PUF.



(B) Enhanced Anderson PUF.

FIGURE 4.5: Global Uniqueness evaluated through the fractional Hamming Distances distributions.

If PUFs output uniformly distributed and independent random response bits, the global uniqueness is close to 50% on average. Values higher or lower than 50% result in lower chip-distinguishability.

Since 165 different PUFs instances were defined, the global uniqueness of the Anderson PUF can be evaluated by calculating $\frac{165\times164}{2} = 13530$ Hamming Distances. Their average yield to a global uniqueness of 47.18%, and Figure 4.5A shows the distribution of the Hamming Distances, which can be assumed to be Gaussian with a mean of 28.30 and a standard deviation of 6.68. It follows that the average response distance is equal to 28.30, which is really close to the ideal value of 30 (all the responses differ from one another for half bits).

The same procedure can be applied on the Enhanced Anderson PUF to evaluate the global uniqueness. However, since there are 2 challenges for each PUF, the number of unique response pairs is equal to $\frac{(165\times2)\times[(165\times2)-1]}{2} = 54285$. Their average yield to a uniqueness of 49.37%, and the Hamming Distance distribution (Figure 4.5B) can be assumed to be Gaussian with with a mean of 29.62 and a standard deviation of 4.82, slightly better than the normal Anderson PUF.

4.1.3.2 Reliability

Ideally, a PUF should always be able to exactly reproduce the same response when the same challenge is applied. However, as anticipated before, since PUFs are based on variations of electrical characteristics, a number of response bits might change under either stable or variable environmental conditions, such as temperature and power supply. To this aim, the reliability metric can be used to estimate the percentage number of bits in a response which does not change value among responses obtained from a repeatedly applied challenge. For a device d, let M be the number of measurements of N-bit responses, $r^{'}_{d,j}$ $(j = 1, 2, ..., m)$, and r_d be the baseline reference response of the d-th device. The reliability can be estimated as:

$$\mathit{Reliability}\,(d) = \left(1 - \frac{1}{M}\sum_{j=1}^{M}\frac{HD(r_d, r^{'}_{d,j})}{N}\right) \times 100\%.$$

A PUF with stable responses achieves a high value of reliability, thus its value should be as close as possible to 100%.

The reliability values of the proposed implementations were calculated both under stable and variable temperature, while applying a fixed voltage of 1.20V. Each PUF response has been sampled 37 times per PUF instance (and with the same applied challenge for the Enhanced PUF), and a temporal majority vote has been applied to determine the most frequent response.

However, even if the temperature on chip is unknown, due to the lack of sensors on board, since the experiments have been conducted in a short time, we can assume that the temperature was stable and equal to the room temperature of about 24°C. Furthermore, the baseline reference response was picked as the most frequent one, and the reliability was calculated as its fractional Hamming Distance with the other 36 responses. The average values of reliability for the 165 PUF instances are 99.85% and 99.91% for the Anderson PUF and the Enhanced architecture, respectively.

Applying the same methodology, it is possible to estimate average reliability values under different conditions. In particular, some experiments were conducted heating the chip for at least 5 minutes with a warm air flow. By exploiting a temperature sensor next to the FPGA die, the temperature was measured and it was equal to about 70°C. Also in this scenario, the baseline reference response was the most frequent one obtained in the previous experiments. Evaluating the fractional Hamming Distance with the other majority responses under the higher temperature value, reliability value for the normal Anderson PUF is equal to 90.79%, while the enhanced architecture achieve the value of

	Global Uniqueness	**Reliability**
Arbiter PUF [66]	23%	99.70%
RO PUF [109]	46.15%	99.52%
Anderson PUF Virtex-5 [8]	48%	96.40%
Anderson PUF Zynq-700 [124]	49.6%	96.3%
Anderson PUF Virtex-6 [54]	N/A	92.97%
Anderson PUF Spartan-6 [45]	N/A	N/A
Anderson PUF Spartan-3E	47.18%	90.79%
Anderson PUF Spartan-3E (Enhanced)	49.37%	98.14%

Table I: Global uniqueness and Reliability estimated for some delay-based PUFs.

98.14%. The enhanced architecture is considerably better than the normal one under temperature variations.

4.1.3.3 Uniformity

A PUF is expected to generate responses containing ideally the same number of logic-0s and logic-1s. Therefore, the uniformity metric (also called randomness by Hori et al. in [53]) can be exploited to estimate the distribution of logic-0 and logic-1 in PUF responses. Let N be the number of response bits, and i be the i-th response, the percentage measure for uniformity of response $r_i = r_{i,1}r_{i,2}...r_{i,N}$ can be defined as:

$$Uniformity\,(i) = \frac{1}{N}\sum_{b=1}^{N} r_{i,b} \times 100\%.$$

A value of 100% means that all r_i response bits are logic-1. For true random bits, uniformity should be as close as possible to its ideal value of 50%. Let R be the number of responses, resulting from the product between the amount of different PUF instances and input challenges (if any). The average uniformity can be calculated as:

$$Uniformity = \frac{1}{R}\sum_{i=1}^{R} Uniformity\,(i)$$

Experimental measurements yield to average uniformity values of 50.59% and 50.99% for the Anderson PUF and the Enhanced architecture, respectively. Being really close to the 50% ideal value, these results confirm once again the good properties of the proposed designs.

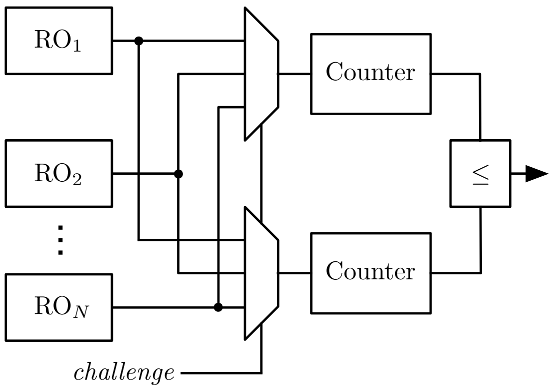


FIGURE 4.6: Ring Oscillator based PUF architecture: a functional overview.

4.2 Ring Oscillator as Secrecy Source for PUFs

RO is a widespread adopted primitive in the hardware design. Thanks to its easiness in the implementation in HDL, it can be used in any hardware technology. In the FPGA design, ROs are mainly exploited to implement secure primitives, such as True Random Number Generator (TRNG) and PUFs. The former is an unstable circuit which has to output a stream of random bits, the latter is a primitive which provides unique and stable hardware fingerprints. Both rely on the RO, but differently, as the TRNG exploits the randomness of oscillations jitter [42, 76] and the PUF exploits the randomness of oscillation frequencies for different ROs [74, 109].

Figure 4.6 illustrates a generic RO PUF architecture. The mechanism exploited by the PUF is a differential frequency measurement operation. Providing a challenge, a pair of ROs is selected and two frequencies are measured by means of counters. Once the frequencies f_a and f_b are gathered, the PUF gives one response bit:

$$r = \begin{cases} 1 & f_a \leq f_b \\ 0 & f_a > f_b \end{cases} \tag{4.2}$$

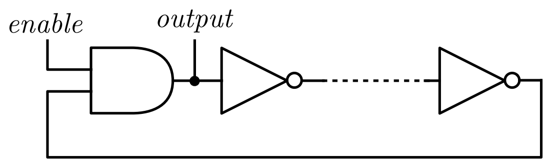
Before try to analyze the RO PUF, this Section introduces a set of experiments which aim is to characterize the frequency behavior. Then, the Subsection 4.3 details a new PUF architecture which is based on ROs.

4.2.1 Research work on RO characterization

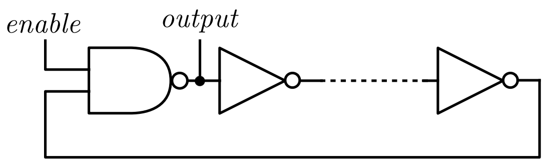
In the literature, a significant amount of research papers have addressed the characterization of ROs and the sensitiveness problem of ROs from external and uncontrolled working conditions. Sedcole et al. in [100] showed a characterization analysis for ROs implemented on 18 Altera Cyclone II FPGAs, demonstrating a systematic process variability. In [75], Maiti et al. characterized the ROs frequencies over a population of 193 Xilinx Spartan-3E S500 FPGAs, giving details about an implementation of RO PUF. Later, the authors of [119] gave a deeper analysis of frequency distribution used in [75]. In particular, through the Anderson-Darling test, similarity analysis and principal component analysis, they demonstrated that ROs are eligible secure primitives, since they are well distributed among different devices and hard to predict. Authors of [92] proposed a technique to mitigate the temperature effect on the RO PUF responses stability. Merli et al. in [77] addressed the problem of the logic which surrounds RO, giving an exhaustive frequencies characterization on a Xilinx Spartan-3E devices, with the aim to define a useful technique to deal with PUF response instability. Amouri et al. analyzed the transistor aging effect on Spartan 6 FPGA devices exploiting the oscillation frequencies from RO [7]. In particular they stressed devices and measured the impact of such a stress on ROs frequencies, demonstrating a degradation of 5.17% on read frequencies.

4.2.2 RO frequencies characterization

The RO PUF is an easily implementable hardware primitive and, with respect to other proposed PUFs architectures, it does not require special attention to symmetric placement, since the RO structure is a single closed loop [109]. For the FPGA technology, this implies a suitable implementation for every device and family. The design parameters which characterize the RO loop are identifiable in: the number of stages, the routing and the placement. As for the first, it affects the oscillation frequency because the increasing of involved stages in the loop causes a greater delay. In the same manner, the routing has to be considered as longer connections cause slower oscillation frequencies. At the end, the placement of the RO loop involves the choice of which basic elements implement the ring stages, i.e., the relative position among loop stages and the position of them within the chip.



(A) Ring oscillator loop controlled by a AND control gate (odd inverters).



(B) Ring oscillator loop controlled by a NAND control gate (even inverters).

FIGURE 4.7: Ring oscillator loop designs.

Other parameters alter the RO frequency, in particular the working conditions of the device. Mainly, they can be considered as unwanted side effects, which cannot be controlled at design level and during the runtime. The supplied voltage is directly related to the signal propagation delay, hence the RO frequency is sensitive to the voltage variations. Those variations can be caused either by unstable supplied voltage or by variable workload of the logic that surrounds the RO, which absorbs a significant current and causes a local voltage drop. Hence, the switching activity, i.e. the logical values switching frequency of the signals of the surrounding logic, represents a disturb.

The die temperature, similarly to the voltage, is able to cause a degradation of the design performance, since at higher temperature values the signal propagation delay increases. Even in this case, two sources can be identified. Obviously the environmental temperature in which the circuit works is responsible for the signal delay over all the die, but a secondary contribution can be caused by local heating. Indeed, a high speed circuitry is able to warm the surrounding area due to dissipation effect, hence it might affect the speed of other on-chip structures.

Another investigated side effect is the aging of the chip. Indeed, even with perfect and stable working conditions, during the chip lifetime, the frequency can be altered by the aging in a permanent manner. In fact, contrary to previous discussed effects, the aging of the chip is incremental and cannot be recovered once happened. Aging has different contributions, such as the hot carrier injection, the oxide breakdown, the electromigration phenomenon, the negative and positive bias temperature instability [68].

4.2.3 RO structure and measurement architecture

The RO structure adopted to characterize the frequencies is reported in Figure 4.7. A control gate interrupts the ring in order to enable or disable the oscillation and its output is exploited to obtain the oscillating signal. If the number of inverting stages are odd, the control gate must be an and-gate (Figure 4.7A), otherwise a nand-gate (Figure 4.7B).

In order to measure the RO frequency value, a simple measure architecture is exploited, similar to the RO PUF case, which requires only two counters. One counter establishes the time window in which the frequency measurement has to be accomplished, hence it is timed by the system clock, which frequency is C, and it is configured to count up to a maximum fixed value T. The other is fed with the RO output, so it counts the edges (rising or falling) of the oscillations. When the first clock reaches the established maximum counting value, the RO is disabled and the RO frequency can be obtained as $R \times C/T$, with R the number of counted oscillations edges. Due to the uncertainty on the system clock phase, this measurement introduces a measurement error $\varepsilon = \pm\frac{C}{2\times T}$.

The measurement architecture is implemented on Xilinx Spartan-6 XC6LX16 45 nm devices within Nexys3 boards. Each CLB of Spartan-6 technology contains a pair of slices, slice L and slice X or slice M and slice X, and all of them are equipped with four 6-input LUTs. The two different slices couples are alternated among CLBs columns [120]. The targeted FPGA device has a CLB array composed of 18 columns and 60 rows.

The clock counter width is 18 bits and the RO counter width is fixed to 24 bits. This choice enables to keep the RO and the two counters bounded in a block of 12 CLB, such that it can be easily placed over allowed FPGA positions. Furthermore, each implemented entity is fixed in position within the cell and in used basic elements. This implies that each cell translation does not alter the design at the netlist level description.

In order to read the bits contained in the RO counter, the design architecture is instrumented with Xilinx Chipscope, in particular with one Virtual IO port and one ICON, and, by adopting the Xilinx software libraries, the value can be elaborated by a PC in which the board is plugged-in (Figure 4.8). Like the RO measurement cell, all involved Chipscope cores are bounded in a fixed shape and constrained in location and basic elements.

4.2.4 Result and validation

This Subsection collects all the experiment and details all the issues involved with measuring technique previously introduced, analyzing read frequencies that are gathered

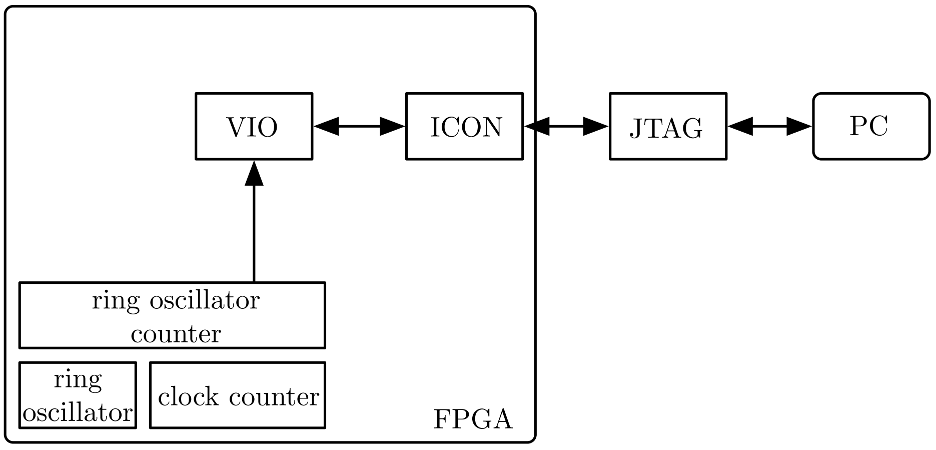
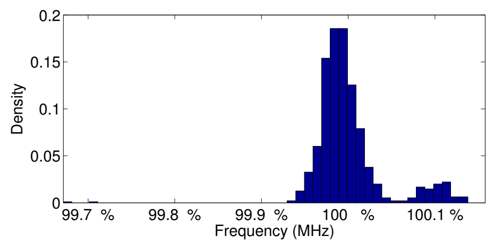


FIGURE 4.8: High level schematic view of the adopted design architecture.

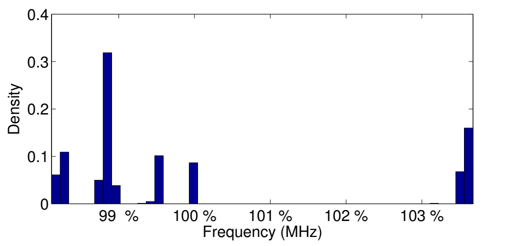
under different conditions. Thus, all the experimental analysis are listed below, in order to illustrate how ROs frequencies are altered by external and uncontrolled conditions.

4.2.4.1 Analysis of the logic which surrounds the RO

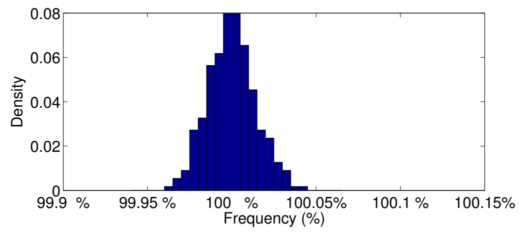
As the surrounding logic is unavoidable to on-chip measure the frequency, it is necessary to evaluate how components surrounding ROs may influence their frequencies. At this aim, first of all the impact of the proximity of both two counters and the Chipscope logic to the RO were evaluated and, to avoid unwanted effects of temperature variations, the external temperature was fixed at 26.6°C by means of a thermal chamber. In particular, tests targeted a single 5-stage RO, implementing it by exploiting several synthesis, changing the on-chip position of the clock counter, RO counter and Chipscope logic, one by one keeping the other fixed. The design diversity allowed to see how the surrounding logic involved in a frequency measurement affects read frequency values. In each experimental campaign ∼1000 experiments were ran and each one was repeated 25 times in order to mitigate the measuring error by averaging the values. Figure 4.9 reports frequencies distributions, obtained by allocating counters and Chipscope in different positions, considered as percentage variations from the average value. As for the RO counter, its position mostly does not influences the frequency value, except for some positions around the same rows in which the RO is placed, causing an increase of 0.1% on read values (Figure 4.9A). Differently, the read frequency is sensitive to the placement of the clock counter, with an alternating of decreases and increases of $-1\%/+\sim 3\%$ (Figure 4.9B). In both the cases, the measured frequencies turned out to



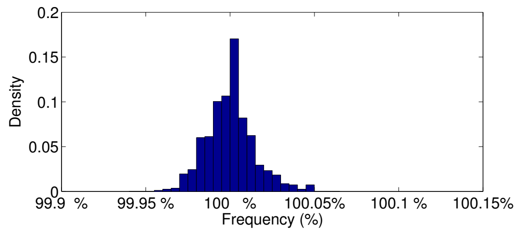
(A) Frequencies distribution moving the RO counter.



(B) Frequencies distribution moving the clock counter.



(C) Frequencies distribution moving the the Chipscope logic bounded in a rectangle.



(D) Frequencies distribution moving the Chipscope logic bounded in a square.

FIGURE 4.9: Distribution of ROs frequencies values, considered as percentage variation from the average, with different places for counters and Chipscope debug logic.

be stable when counters were placed close to ROs. Figures 4.9C and 4.9D show that the impact of Chipscope on read frequencies is practically insignificant, even changing the shape in which its logic is bounded. In particular, placing Chipscope logic in different vertical positions does not have any significant effect, but moving it horizontally causes a slightly frequencies decrease (maximum ∼0.05%) proportionally to the distance. Hence Chipscope can be considered as a non-intrusive surrounding logic. Indeed, during the oscillations sampling process, its logic does not work.

To better evaluate the effect of an intrusive logic that heavily works near the RO, an architecture characterized by a pseudo-random behavior was designed. It is inspired by the Linear Feedback Shift Register (LFSR). Compared to a classic LFSR, which is a single shift register which combines some values with a xor function to drive the shift-in input, the surrounding logic was defined to perfectly fit the slices structure, guaranteeing a higher workload. Since each slice contains four 6-input LUTs and their outputs can be registered in flip-flops, a very pervasive surrounding logic has to exploit all LUTs inputs with high switching activity signals and has to occupy as much as resource in the slices, including registers. Figure 4.10 shows an high level schematic of the devised logic. Each CLB has two slices which generate pseudo-random switching activity exploiting four parallel 6-input xor functions and storing the generated output values in flip-flops. The input assignment for xor function involves four signals locally picked, i.e. within the

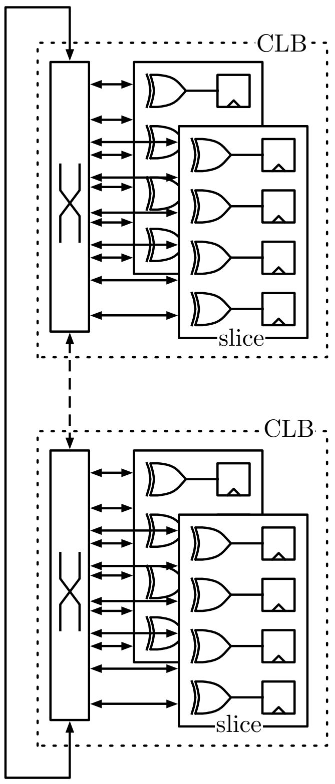
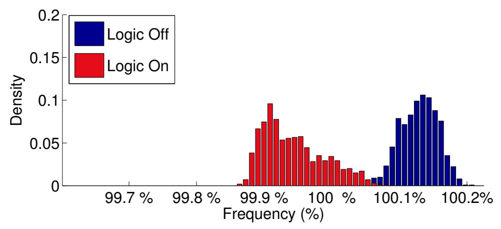


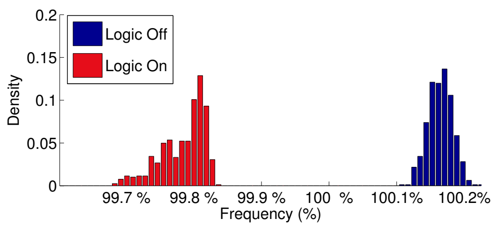
FIGURE 4.10: A schematic overview of the implemented intrusive logic, which fit the internal structure of the Spartan-6 device.

CLB, and two from outside signals of neighbor CLBs. Iterating such structure in a loop generates an auto-sustained signal switching, like the LFSR, but with more simultaneous activities per clock-cycle.

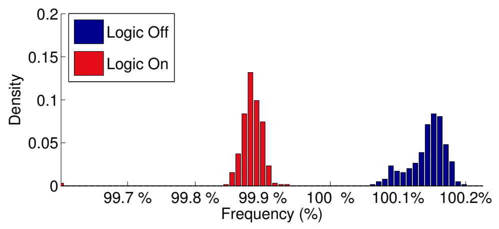
The cells activity can be easily disabled driving the signal clock-enable for all the involved flip-flops. The density of this intrusive logic, evaluated as the the number of occupied LUTs on four times the number of occupied slices, reaches values between 75% and 85%. Figure 4.11 reports three experiments with different intrusive logic configurations, respectively 3×8, 8×6 and 6×18, considering the frequency as percentage against average values of targeted RO measured without any insertion of intrusive logic. In all reported cases the logic causes a frequency decreasing around 0.4% when it is on and, surprisingly, it causes a frequency increasing of about 0.15% when it is off. Moreover, the logic is more intrusive in frequencies measurements when the prominent dimension is the height, hence the logic turns out to be more invasive if it is vertically stretched.



(A) Frequencies distribution considering a shape of 3×8 CLB.



(B) Frequencies distribution considering a shape of 8×6 CLB.



(C) Frequencies distribution considering a shape of 6×18 CLB.

FIGURE 4.11: Distribution of ROs frequencies values with an intrusive surrounding logic, considered as percentage variation from the value of ROs without the logic.

4.2.4.2 Analysis of the stages number and routing

The frequency value of ROs is thightly coupled with the number of stages in the ring, such that longer loops cause lower frequencies. In order to study the frequency behavior in a statistical manner, a huge quantity of frequencies were extracted from different locations on chip and by exploiting several chips. Let $f_{n,m,d,s}$ be the frequency read from the n-th RO at the m-th experiment with s inverting stages placed on the d-th device. Then, let N, M D and S be respectively the number of ROs that have been placed on chip, the number of trials to sample the same RO, the number of involved devices and the set of the different defined stages. For the sake of ease, let $f_{n,d,s}$ be the mean frequency value obtained averaging all the k trials:

$$f_{n,d,s} = \frac{1}{M}\sum_{m=0}^{M-1} f_{n,m,d,s}. \tag{4.3}$$

For each s it is possible to calculate the associated average frequency value:

$$f_s = \frac{1}{D}\sum_{d=0}^{D-1}\frac{1}{N}\sum_{n=0}^{N-1} f_{n,d,s} = \frac{1}{D}\sum_{d=0}^{D-1} f_{d,s} = \frac{1}{N}\sum_{n=0}^{N-1} f_{n,s} = \frac{1}{D\cdot N}\sum_{d=0}^{D-1}\sum_{n=0}^{N-1} f_{n,d,s}\forall s\in S.$$

Then, the global standard deviation is defined as:

$$\sigma(s) = \sqrt{\frac{\sum_{d=0}^{D-1}\sum_{n=0}^{N-1}\left(f_{n,d,s} - f_s\right)^2}{D \cdot N}}, \forall s \in S.$$

Other standard deviations can be computed, considering the distribution of frequencies within the same device and among different devices. To this aim, the intra-die standard deviation is defined as:

$$\sigma_{\text{intra}} = \sigma_{d,s} = \sqrt{\frac{\sum_{n=0}^{N-1}\left(f_{n,d,s} - f_{d,s}\right)^2}{N}}, \forall s \in S, \forall d. \tag{4.4a}$$

Contrary to the intra-die, the inter-die standard deviation has to be calculated among different devices, hence:

$$\sigma_{\text{inter}} = \sigma_{n,s} = \sqrt{\frac{\sum_{d=0}^{D-1}\left(f_{n,d,s} - f_{n,s}\right)^2}{D}}, \forall s \in S, \forall n. \tag{4.4b}$$

Five different loops were defined: their number of inverting stages was in the range $[4, 8]$. In each design, the control gate of the loop was fixed in the bottom LUT of the slice X and the loop has been arranged in the other available LUTs in slice X and the other slice, which kind (M or L) depends on the CLB column. Each RO measurement block is implemented in every allowable place on the FPGA, as in the previous experimental campaigns, obtaining about 1000 different design implementations, hence ~1000 frequency values. Ten devices were involved in the measurement campaign. The Table II lists mean and standard deviation values of frequencies for the five configurations.

The higher is the amount of stages, the lower are both average values and associated standard deviations. This implies that if the loop is longer, the values are closer to the average frequency values. Indeed, since each loop stage introduces a delay that is affected by uncertainty due to manufacturing variations, the uncertainty on the global delay turns out averaged.

The same happens also considering standard deviations of frequencies among different devices. In fact, in the same table there are two other standard deviations varying the

RO Stages	Mean (MHz)	Standard Deviation (MHz)[1]		
		Global	Intra-die	Inter-die
4	351.4552	7.1784	1.5295	2.5168
5	347.3760	7.0729	1.0850	2.0569
6	259.1042	5.1403	0.9113	1.4903
7	201.2785	3.6479	0.5729	0.9840
8	190.6770	3.7693	0.5320	0.8398

TABLE II: Mean values and standard deviations of RO frequencies for different stages. The Intra-die and Inter-die are calculated among 10 ROs and 10 devices.

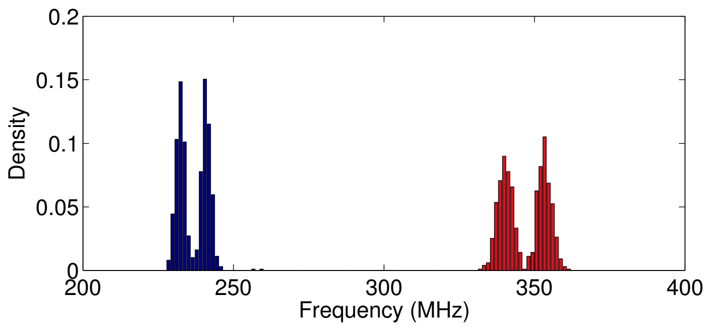


FIGURE 4.12: Frequency distribution for two different mapping 4-stages RO configurations placed over all the FPGA device.

number of RO stages: one is the average value, among 10 devices, of standard deviations calculated considering 10 different ROs, i.e., average intra-chip standard deviation reported in Equation 4.4a; the other is the average value, among 10 ROs, of standard deviations calculated for 10 different devices, i.e., average inter-chip standard deviation reported in Equation 4.4b. These two quantities represent how scattered are frequencies and, since intra-chip dispersion is greater than inter-chip ones, they illustrate that frequencies are closer among them when they are picked from the same device than the case in which they are from different devices. Moreover their dispersion values are inversely proportional to the number of stages.

Furthermore, besides the RO structure, the placement and routing of the loop plays a crucial role in determining the oscillating frequencies. In order to put in evidence the relationship among different routing configurations and corresponding frequencies, two identical 4-stages ROs were designed, but with different mapping within a slice X. In particular, the architectures were obtained by swapping two stages in the LUT assignment such that involved paths were different at least for two stages. The two

[1]As explained later, the RO frequency characterization is different considering RO implemented in CLBs of even columns and odd columns. Standard deviations reported in table are evaluated considering both the frequency distributions associated to odd and even columns. The experimental values for each frequency group are close to $\sigma/\sqrt{2}$, since for each kind of CLB the standard deviations are very similar.

configurations were placed in every allowable location on the FPGA and their frequency distributions are reported in Figure 4.12. The average values differ from one another by ∼100 MHz and their standard deviations by ∼2.35 MHz. Moreover, each distribution contains two well-distinguishable frequencies peaks. Correlating the frequencies with the spatial position, it is possible to note that ROs which are placed within a CLB in even columns, characterized by a slice X and slice L pair, have frequencies that in values are less than others placed in odd columns, characterized by a slice X and slice L pair. This happens even if the RO structure is entirely placed within a slice X. We can conclude that the two different distribution peaks are caused by different involved routing resources.

4.2.4.3 Temperature analysis

The working temperature for an integrated circuit is directly related to the signal propagation delay. Indeed, high working temperatures cause a performance degradation in terms of speed. In order to analyze the effect of temperature changing on ROs frequencies, 5 stages ROs were characterized on a single device under 6 different external fixed temperature: 0°C, 13.3°C, 26.6°C, 40°C, 53.3°C and 66.6°C. To this aim, the FPGA was placed in a thermal chamber which keeps the temperature with a precision of 0.1°C. When the thermal chamber reached the right temperature, the frequency measurement process have started after 30 minutes in order to be sure that the die has uniformly reached the same external temperature before starting each test campaign. Figure 4.13 illustrates all measured frequencies varying the temperature within the range $[0°C, 66.6°C]$. They are inversely proportional to temperature values and the relationship between them is quite close to be a linear function. The only observable exception is after 40°C because the curve start to be more descendant. To better analyze the relationship between the temperature value and the ROs frequencies, we can consider the difference quotient for each temperature range, namely how the frequency decreases increasing the temperature of 13.33°C.

Temperature (°C)	Even Column		Odd Column	
	Frequency (MHz)	Difference Quotient	Frequency (MHZ)	Difference Quotient
0	348.5540		361.8935	
13.3	344.7655	-0.2842	357.9787	-0.2937
26.6	340.8063	-0.2970	353.8691	-0.3083
40	336.8726	-0.2951	349.7988	-0.3054
53.3	332.4376	-0.3327	345.1964	-0.3453
60	327.7191	-0.3540	340.2945	-0.3677

TABLE III: Frequencies and difference quotients for ROs placed in even and odd columns varying the working temperature.

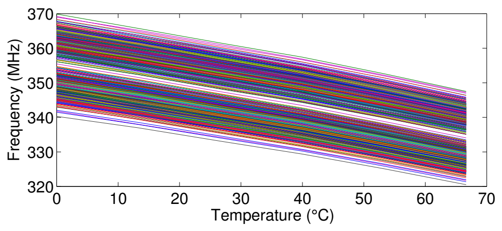


FIGURE 4.13: Values for all ROs frequencies varying the working temperature.

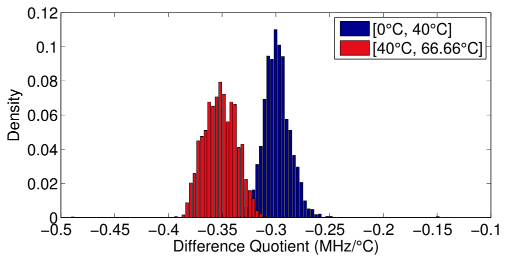


FIGURE 4.14: Difference quotient distributions evaluated before and after 40°C.

Figure 4.14 shows two distributions difference quotients calculated for all ROs. The blue histogram is related to the temperatures less than 40°C, while the red to the temperature greater than 40°C. They indicates that the average values of difference quotients are respectively -0.29 MHz/°C and -0.36 MHz/°C. Both are distributed with a standard deviation of ∼0.013 MHz/°C.

As for the even and odd CLB columns, there is a difference in terms of difference quotient, as illustrated in Table III. The distance between the difference quotient of even and odd columns can be approximated as constant and equal to 0.015 MHz/°C. This implies that the frequencies of ROs placed in even columns are more sensitive than others placed in odd columns.

4.2.4.4 Aging analysis

The aging is an unavoidable process that afflicts any IC, causing performance degradation and leakage current increase. In order to evaluate its impact on ROs, it is possible to perform aging acceleration through stressful working conditions, in particular with high temperature and supplied voltage. At this aim, the FPGA core was stressed with

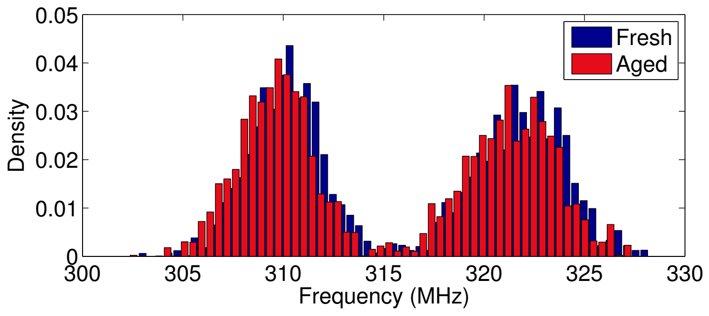


FIGURE 4.15: Frequency distributions measured on the same device before and after the aging process.

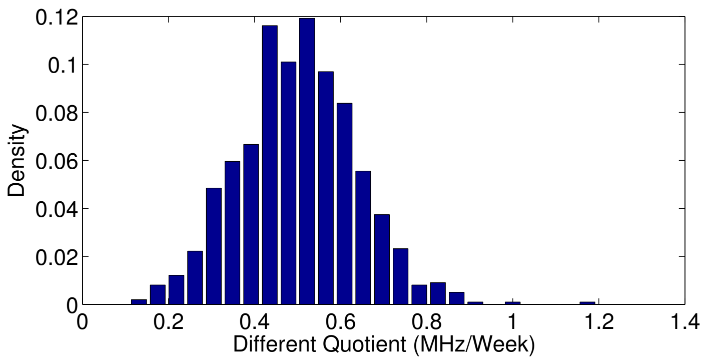


FIGURE 4.16: Frequency distributions measured on the same device before and after the aging process.

an external power supply at 1.8 V (+50% more than the nominal value) and heating the chip up to 80°C for 7 days. In order to clear the effects of the reversible aging process, the FPGA was recovered in 3 days under nominal working conditions but at 80°C. As in the temperature experiments, a 5 stages ROs was adopted to characterize the device before and after the aging process. Figure 4.15 illustrates such characterization through frequency histograms. The aged device frequency distribution has the same shape as the fresh version and it is shifted by ∼0.6 MHz. Indeed, as reported in Figure 4.16, the difference quotient values are distributed as a gaussian which means is around 0.6 MHz.

4.3 Frequencies Signature PUF

This Section introduces a new technique to recognized a device by exploiting ROs. Contrary to available RO PUF architectures, which compares two ROs to extract a 1-bit response, this technique considers directly frequency values to discriminate several silicon devices. For this reason, the devised PUF is defined as Frequencies Signature

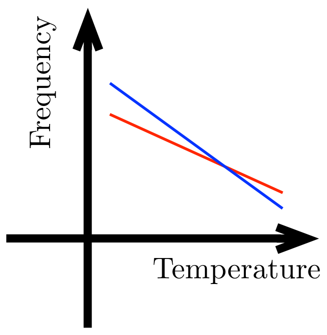


FIGURE 4.17: Temperature (but also other working parameters) can differently affect two RO, causing unstable responses for the RO PUF.

based PUF (FS PUF). The FS PUF does not provide sequences of bits, but a mechanism to authenticate silicon devices. Such mechanism, applying some transformations to the read frequencies, is inherently immune to external changes. The FS PUF effectiveness it is mathematically demonstrated and empirically evaluated trough experimental results.

4.3.1 A model of read frequencies

Experiments conducted and illustrated in Subsection 4.2.4 put in evidence that, for each RO, measured frequencies are distributed as a gaussian $\mathcal{N}\left(\mu, \sigma^2\right)$, which mean and standard deviation can be estimated with some frequency samples. Considering the Equation 4.2, when the distributions associated to the selected frequencies are characterized by two mean values very similar or, in general, by two gaussian distributions which have a significant frequency values in common, the stability of such a pair turns out to be very low. Furthermore, if this situation does not seem to be possible at normal working conditions, with external disturbances it might happen. For instance, with regards to the temperature variations, two frequencies generated by two ROs can be differently modified because of their difference quotients, causing a reversing of their mutual order, as pictured in Figure 4.17.

Ideally, the RO PUF has to work with frequencies which values are stable and deterministic during the time. But there are some unavoidable uncertainty sources, mainly caused by the electric phenomenon itself and by the measurement architecture. The mere RO frequency value, of course, depends on the manufacturing variations and on the working conditions. We can assume that a read frequency f is a quantity characterized as

RO Stages	Measurement Error (Hz)
4	136,137.27
5	124,100.96
6	94,084.62
7	80,469.53
8	77,047.38

TABLE IV: Measurement error ε_s associated with different stages.

follows:

$$f_{n,m,d,s} = \widehat{f}_{n,m,d,s} + \varepsilon_{n,m,d,s} + t_{n,m,d,s}. \tag{4.5}$$

In the equation, $\widehat{f}$ is the inherently RO frequency, ε is the error associated with the measure and t is a quantity given by the uncontrolled operational conditions, such as the temperature.

As for the $\widehat{f}$, previous experimental campaigns demonstrates that it is a quantity distributed as a gaussian curve (Subsection 4.2.4). Within the same device, the associated standard deviation $\sigma_{d,s}$ is always less than the one considering several devices, i.e. $\sigma_{n,s}$, as reported in Table II. Moreover, such quantities decrease with the stages' growing.

The measurement operation introduces an error ε, which is random and cannot be directly controlled, but at least can be estimated. We can assume that it is an additive white gaussian noise (AWGN), hence its values are distributed all around the 0: $\varepsilon \sim \mathcal{N}\left(0, \sigma_\varepsilon^2\right)$. Indeed, considering that ε afflicts all the measurement $f_{n,m,d,s}, \forall n, m, d, s$ (see Subsection 4.2.4.2), its standard deviation can be calculated as:

$$\sigma_{\varepsilon_{n,d,s}} = \sqrt{\frac{\sum\limits_{m=0}^{M-1} \left(f_{n,m,d,s} - f_{n,d,s}\right)^2}{M}}.$$

In the equation, $f_{n,d,s}$ is the average value over M trials (see Equation 4.3). Then, σ_{ε_s} can be globally estimated over all the done experiments as average value:

$$\sigma_{\varepsilon_s} = \frac{1}{D}\sum_{d=0}^{D-1}\frac{1}{N}\sum_{n=0}^{N-1}\sigma_{\varepsilon_{n,d,s}} = \frac{1}{D\cdot N}\sum_{d=0}^{D-1}\sum_{n=0}^{N-1}\sigma_{\varepsilon_{n,d,s}}.$$

Table IV reports the values of ε_s against different number of stages. Like other standard deviations reported in Table II, it decreases with the growth of the RO loop stages.

As one can notice, there is a trade off given by the choice of the number of stages, the associated measurement error and the frequency dispersion. Indeed, the best situation for the PUF based on ROs is a high frequency dispersion, in order to have better uniqueness, and low measurement error, in order to reach better stability of measured values. This implies that with a huge population, which requires a very widespread frequencies range, a low number of stages should be picked. Hence, to face with huge measurement error, a significant number of trials (i.e. measurements) has to be accomplished in order to mitigate the contribution of the error. Otherwise, with a small population, the number of stages could be higher, with a small error value. Hereafter, all the definitions and experiments will be given considering a 5-stages RO and the subscript s will be no more considered. This assumption simplifies all the further discussions.

At the end, as for the t in the Equation 4.5, this quantity can be considered as a shift for the frequency value. Indeed, external uncontrolled working conditions, such as temperature and voltage, cause a shift related to the expected value. Such shift is strictly coupled with the frequency reference. For instance, the reference frequency should be measured under fixed conditions of temperature (e.g. 25°C), voltage (e.g. 1.2V), etc. For the scope of the FS PUF, t does not need to be defined in someway, since, as shown in the next Subsection, it is not considered as part of the signature.

4.3.2 Frequencies as a signature

This Subsection introduces the device signature obtained as collection of some measured frequencies. With this goal, let $S_{m,d}$ be the frequencies associated to the d-th device; m indicates a single measurement among different trials, accomplished in order to have more than one signature for each device. To better clarify, m can be considered as a discrete time variable, since $S_{m,d}$ and $S_{m',d}$ are collected from the same device, but in two different instants. Formally, $S_{m,d}$ is a N dimensional vector defined as follows:

$$S_{m,d} = \{f_{n,m,d}\}, \forall n \in [0, \dots, N-1] \rightarrow S_{m,d} = \{f_{0,m,d}, f_{1,m,d}, \dots, f_{N-1,m,d}\}.$$

Substituting each component $f_{n,m,d}$ with the definition 4.5, the signature definition can be expressed as a sum of three vectors:

$$S_{m,d} = \{f_{n,m,d}\} = \left\{\widehat{f}_{n,m,d}\right\} + \{\varepsilon_{n,m,d}\} + \{t_{n,m,d}\}, \forall n \in [0, \dots, N-1]. \tag{4.6}$$

In particular, from 4.6 it is clear that each n-th component has a frequency value $\widehat{f}_d$, which is characterized by a known mean and, since $S_{m,d}$ is extracted within the same device, by a known (intra-die) standard variation, both given by the Table II considering 5 stages, $\widehat{f}_d \sim \mathcal{N}\left(\mu_{f_d}, \sigma_d\right)$. Furthermore, such frequency value does not depend on the m variable, as at each trial the value is the same. In other words, each RO instance during the manufacturing process is characterized by $\widehat{f}_{n,d}$ as a fixed value of a gaussian distribution (previously defined) and does not change its inherently value during the time. Indeed, as reported by the model in the Equation 4.5, $\widehat{f}_{n,d}$ is not the actual oscillation frequency, but it has to be considered together with the quantity t.

As for the measurement error, each n-th component of $S_{m,d}$ is affected by a value $\varepsilon_{n,m,d}$. We can assume that the distribution of ε does not depend on the specific device, trial or RO, since the error is given by frequency measurement process itself. At each measurement it can be considered as a value of an AWGN process. Moreover, ε has the same behavior averaged over time ($\underset{m}{E}\left[\varepsilon_{n,m,d}|n,d\right]$), averaged over the realizations ($\underset{n}{E}\left[\varepsilon_{n,m,d}|m,d\right]$) and averaged over the devices ($\underset{d}{E}\left[\varepsilon_{n,m,d}|n,m\right]$). In fact, there are no differences in measuring from another RO the frequency value or repeating from the same RO another frequency measurement, since the conditions on all the cases are the same. Of course, the measurement design has to be symmetric with the respect to ROs and the same for all the devices. Hence, formally:

$$\underset{m}{E}\left[\varepsilon_{n,m,d}|n,d\right] = \underset{n}{E}\left[\varepsilon_{n,m,d}|m,d\right] = \underset{d}{E}\left[\varepsilon_{n,m,d}|n,m\right] = 0. \tag{4.7}$$

The equation 4.7 is valid for a significant instances of ε values, hence:

$$\underset{m}{E}\left[\varepsilon_{n,m,d}|n,d\right] = \frac{1}{M}\sum_{m=0}^{M-1} \varepsilon_{n,m,d} \xrightarrow{M\to\infty} 0. \tag{4.8a}$$

$$\underset{n}{E}\left[\varepsilon_{n,m,d}|m,d\right] = \frac{1}{N}\sum_{n=0}^{N-1} \varepsilon_{n,m,d} \xrightarrow{N\to\infty} 0. \tag{4.8b}$$

$$\underset{d}{E}\left[\varepsilon_{n,m,d}|n,m\right] = \frac{1}{D}\sum_{d=0}^{D-1} \varepsilon_{n,m,d} \xrightarrow{D\to\infty} 0. \tag{4.8c}$$

As anticipated before, t defines, for each component, a frequency shift due to operational conditions. Experimental results discussed in Subsection 4.2.4.3 and 4.2.4.4 highlight that the difference quotients are not the same for every RO, but they are distributed as gaussian curves. Consequently, even if under the same working conditions, t is a quantity that slightly differs among the components of the signature $S_{m,d}$. The assumption that all the t_n are equal for each component causes a small error (compared to the error value), hence hereafter $t_m = t_{n,m}, \forall n \in [0, \dots, N-1]$. Indeed, this substitution is able to add an error of maximum $\sim$1 MHz and with a very low probability[2].

With the previous consideration, $S_{m,d}$ can be simplified:

$$S_{m,d} = \left\{\widehat{f}_{n,d}\right\} + \{\varepsilon_{n,m,d}\} + t_m.$$

Interestingly the average value of defined $S_{m,d}$ components has the following form:

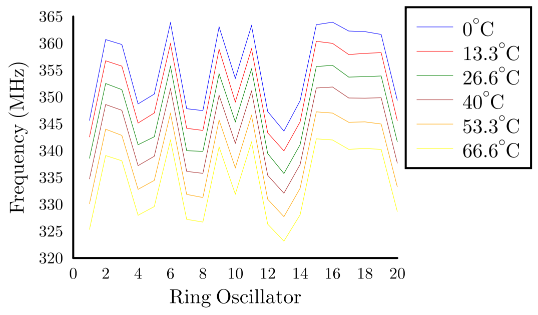
$$\underset{n}{E}\left[S_{m,d}|m,d\right] = \underset{n}{E}\left[\widehat{f}_{n,d}|m,d\right] + \underset{n}{E}\left[\varepsilon_{n,m,d}|m,d\right] + \underset{n}{E}\left[t_m|m,d\right] \approx \mu_{f_d} + t_m.$$

In particular, such approximation is better with greater values of N (Equation 4.8b), since the involved average error is only an estimator. It is worth noting that $\underset{n}{E}\left[S_{m,d}|m,d\right]$ contains the value t_m, which can be exploited to remove it from each component of $S_{m,d}$. Let $\widetilde{S}_{m,d}$ be the signature obtained by subtracting from each component of $S_{m,d}$ the average value $\underset{n}{E}\left[S_{m,d}|m,d\right]$:

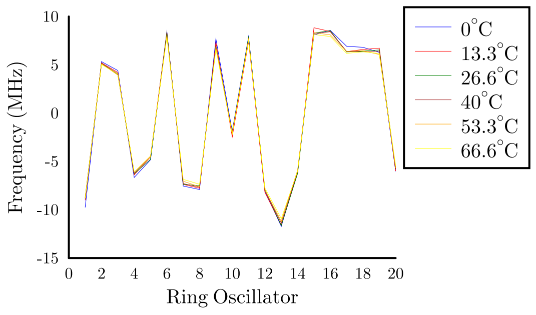
$$\begin{aligned}\widetilde{S}_{m,d} &= S_{m,d} - \underset{n}{E}\left[S_{m,d}|m,s\right] \approx \left\{\widehat{f}_{n,d}\right\} - \mu_{f_d} + \{\varepsilon_{n,m,d}\} \\ &= \left\{\widetilde{f}_{n,d}\right\} + \{\varepsilon_{n,m,d}\}, \forall n \in [0, \dots, N-1].\end{aligned}$$

Consequently, $\widetilde{S}_{m,d}$ does not depend on the working conditions and each frequency component has a new value, given by $\widehat{f}_{n,d} - \mu_{f_d} = \widetilde{f}_{n,d}$. In particular, $\widetilde{f}_{n,d}$ now is characterized by the gaussian distribution of $\widehat{f}_{n,d}$, but the mean value is close to 0 by definition. Such distribution depends only on manufacturing variations and each value $\widetilde{f}_{n,d}$ is related exclusively on it. Furthermore, $\widetilde{f}_{n,d}$ depends on the selected ROs and, hence, on the number of ROs (N) in the signature, because picking different ROs changes the value μ_{f_d}. Consequently, it is not possible to extract the pure $\widehat{f}_{n,d}$, but at least $\widetilde{S}_{m,d}$ contains an expression of the measured frequencies which is not influenced by working conditions. Surely, for higher values of N, μ_{f_d} becomes more close to real

[2]The hypothesis in t introduced the maximum error for each component when the different quotient of the considering RO is very far from the mean value of the distribution. Being a gaussian, this situation is unlikely.



(A) Linear plot of $S_{m,d}$ with 20 ROs frequencies at different temperature.



(B) Linear plot of $\widetilde{S}_{m,d}$ with 20 ROs frequencies at different temperature.

FIGURE 4.18: Comparison between $S_{m,d}$ and $\widetilde{S}_{m,d}$.

average frequency, hence components of $\widetilde{S}_{m,d}$ are less susceptible to changes of N and of ROs.

Figures 4.18A and 4.18B illustrate $S_{m,d}$ and $\widetilde{S}_{m,d}$ respectively, characterized by 20 different ROs and 6 trials made at different temperature. $\widetilde{S}_{m,d}$, as observed from the Equation 4.9, is free of any shift caused by external conditions, even if for some RO a considerable residual error can be observed, partially given by the ε values. Moreover, all the components are distributed around 0, as a consequence of the subtraction of μ_{f_d}.

4.3.2.1 $\widetilde{S}_{m,d}$ in hardware

Having the PUF mechanism implemented in hardware guarantees a stronger security than in the case when some software modules are involved, such as in the case of noisy responses which have to be recovered by a fuzzy extractor (see Subsection 2.2.2). Indeed, as introduced in Chapter 2, the software trustworthiness relies on the hardware, which has to guarantee protection against attacks. If the PUF is one of the mechanisms which cooperates to enable the system security, the software needs to be trusted in someway, with additional precautions, e.g. the chain of trust.

Since the main goal of this Chapter is to give a PUF which can be available on old and low-end FPGA manufactured without any built-in mechanism, $\widetilde{S}_{m,d}$ has to be available directly in hardware. Once the frequencies are read from the ROs, to obtain $\widetilde{S}_{m,d}$ it is necessary to calculate the average value of $S_{m,d}$ and subtract it to each frequency component of the signature. As for the average value, it involves the addition of all the frequency and a division by N, that is a natural number. To avoid the adoption of a complex division algorithm, which requires a significant amount of resources to be accomplished in hardware, the FS PUF can be designed with a signature $S_{m,d}$ which number of components N is a power of 2: $N = 2^x$. Indeed, the evaluation of the mean from 2^x binary values requires to add them without considering the x least significant bits, since the division by a power of two, induced by the mean, is equivalent to a right shift. The possible loss of precision caused by the shifted-out bits is not critical, since they represent the decimal part of the mean value against very higher values.

Therefore, in order to obtain $\widetilde{S}_{m,d}$ in hardware, the only hardware operations required are the addition and the subtraction.

4.3.3 FS PUF: signatures comparison

As anticipated before, the aim of the FS PUF is not to provide response bits, as in common PUF architectures (e.g. the Anderson PUF, illustrated in Section 4.1). The scope of the FS PUF is to provide an effective method to authenticate devices by means of measured frequencies from ROs. Given a test procedure ξ and two signatures $S_{m,d}$, the authentication mechanism has to work as follows:

$$\xi\left(\widetilde{S}_{m,d}, \widetilde{S}_{m',d'}\right) = \begin{cases} 1 & d = d', \forall m, m' \\ 0 & d \neq d', \forall m, m' \end{cases} \tag{4.9}$$

Furthermore, ξ has to be easy to evaluate and to implement in hardware. As stated before, this enables to embed not only the FS PUF within a device, but also the comparison mechanism without occupying too much resources.

4.3.3.1 Distance metrics

There are many available metric functions which are able to compare two vectors to measure how similar are them. For instance, the euclidean distance or the angles between two vectors could be candidates as ξ function. But, first of all, they are not easy to compute in hardware, since they requires complex arithmetic operations and could involve also floating point units. Then, their outputs are not binary, as represented by the Equation 4.9 and need for further evaluations to decide if two signatures are generated from the same devices. Indeed, once computed, the euclidean distance and the angle formed by vectors give a number that has to be compared with a threshold. If the distance is under the threshold or if the angle is in a given range, the signatures are enough similar, then they are classified as provided by the same device. The threshold definition is not an easy task and even not so effective, since the choice of such a value affects all FS PUFs instances effectiveness. For instance, let $\left\|W_{d,d'}\right\|$ be the euclidean norm (L^2 norm) of the vector obtained by subtraction of the corresponding components of $\widetilde{S}_{m,d}$ and $\widetilde{S}_{m',d'}$:

$$\left\|W_{d,d'}\right\| = \left\|\widetilde{S}_{m,d} - \widetilde{S}_{m',d'}\right\| = \left\|\left\{\widetilde{f}_{n,d}\right\} + \{\varepsilon_{n,m,d}\} - \left\{\widetilde{f}_{n,d'}\right\} - \{\varepsilon_{n,m',d'}\}\right\|$$
$$= \begin{cases} \sqrt{\sum\limits_{n=0}^{N-1} \left(\varepsilon_{n,m,d} - \varepsilon_{n,m',d'}\right)^2} & d = d', \forall m, m' \\ \sqrt{\sum\limits_{n=0}^{N-1} \left[\left(\widetilde{f}_{n,d} + \varepsilon_{n,m,d}\right) - \left(\widetilde{f}_{n,d'} + \varepsilon_{n,m',d'}\right)\right]^2} & d \neq d', \forall m, m' \end{cases}$$

The corresponding ξ function obeys to this inequality:

$$\xi\left(\widetilde{S}_{m,d}, \widetilde{S}_{m',d'}\right) = \left\|W_{d,d'}\right\| \leq \tau.$$

The goal of ξ is to discriminate if $W_{d,d'}$ contains only measurement error ($d = d'$) or not ($d \neq d'$), hence τ has to be defined such that it is greater than all the $W_{d,d'}$ with $d = d'$ and less than $W_{d,d'}$ with $d \neq d'$. It is easy to imagine that this bi-partition of $W_{d,d'}$ values could not be easily accomplished, implying a high number of false positives ($\widetilde{S}_{m,d}$ is recognized as a signature generated by another d' device) with a more permissive threshold or false negatives ($\widetilde{S}_{m,d}$ is not recognized as a signature provided by the device d itself) with a less permissive threshold. Moreover, to avoid a high false rejection rate (FRR) in this case, τ has to be fixed at the worst case value, that is the case in which measurement errors on each component are maximum:

$$\tau = \sqrt{\sum_{n=0}^{N-1} \max_{m,d,m',d'} \left[\left(\varepsilon_{n,m,d} - \varepsilon_{n,m',d'}\right)^2\right]}.$$

Such defined threshold generates high false acceptance rate (FAR) because it includes with high probability also signatures which distance is under the threshold due to proximity of some homologous measured frequencies. The FAR can decrease with a higher number of instantiated ROs (N) such that the probability to generate false positives becomes smaller. The same discussion is still valid for other tests which exploit some kind of distance, but are not reported here for sake of brevity.

4.3.3.2 Score test

The threshold selection illustrated before is a critical operation, since it affects the FS PUF instances in terms of FAR and FRR. In particular, this happens because the threshold is applied to a single value given by a metric (e.g. norm or angle). Splitting the signatures in L sub-parts enables to compare them with more than one test and threshold. Assuming that each comparison gives 0 or 1, as illustrated before in the Equation 4.10, all the tests establish a score in the range $[0, L]$. A score of L implies a high similarity between $\widetilde{S}_{m,d}$ and $\widetilde{S}_{m',d'}$, vice-versa a score of 0 implies a high dissimilarity. Contrary to the case of only one comparison test, the choice of the threshold with L tests is less critical because comparisons results are masking by the score. Changing different thresholds affect a subset of L tests; indeed, scores are lower with less permissive thresholds, or higher with more permissive values. The minimum score required to recognize two signatures as generated from the same device can be tuned in order to obtain good FAR and FRR values.

Considering $L = N$ and a unique threshold in each test, comparisons take into account all homologous components of each vector with one test. Hence, with N comparisons between components of $\widetilde{S}_{m,d}$ and $\widetilde{S}_{m',d'}$, the signatures can be considered as provided by the same device if at least l of them succeed. Formally, let χ be the binary function which compares two real numbers and outputs 0 or 1:

$$\chi : \mathbb{R} \times \mathbb{R} \to \{0,1\} \,|\chi(a,b) = \begin{cases} 1 & \text{if } |a| \leq b \\ 0 & \text{if } |a| > b \end{cases}$$

Then, ξ can be defined as follows:

$$\xi\left(\widetilde{S}_{m,d}, \widetilde{S}_{m',d'}\right) = \chi\left(l, \sum_{n=0}^{N-1} \chi\left(\left(\widetilde{f}_{n,d} + \varepsilon_{n,m,d}\right) - \left(\widetilde{f}_{n,d'} + \varepsilon_{n,m',d'}\right), \tau\right)\right), l \in [0, \ldots, L]\,. \tag{4.10}$$

This form of ξ exploits the score obtained by comparing each component of the signature $\widetilde{S}_{m',d'}$ against $\widetilde{S}_{m,d}$ (or conversely $\widetilde{S}_{m,d}$ against $\widetilde{S}_{m',d'}$) with a threshold. Hence, if at least l tests succeed, the signatures are classified as provided by the same device. Furthermore, such ξ function requires only subtractions and comparisons, suitable to be implemented in hardware without adopting any software module.

Statistical model of the score test The effectiveness of the score test, given in the Equation 4.10, relies on the choice of an appropriate threshold τ and minimum score l. Such parameters can be retrieved from a statistical model of the test introduced by ξ over a set of signatures.

The function ξ can me modeled as a Bernoulli process, in which each test performed by $\chi\left(\left(\widetilde{f}_{n,d}+\varepsilon_{n,m,d}\right)-\left(\widetilde{f}_{n,d'}+\varepsilon_{n,m',d'}\right),\tau\right)$ has a value of either 0 or 1 and for each n the probability that $\chi=1$ is p. Each comparison is memoryless, so all evaluations of χ are independent. This assumption relies on the independence of each measured frequency, hence on the FS PUF design that has to properly retrieve frequencies from ROs. In other words, given that the probability p is known, $n-1$ χ tests do not provide any additional information about the n-th χ.

As for p, the probability that $\chi=1$ can be defined as follows:

$$\begin{aligned}p &= P\left(\chi\left(\left(\widetilde{f}_{n,d}+\varepsilon_{n,m,d}\right)-\left(\widetilde{f}_{n,d'}+\varepsilon_{n,m',d'}\right),\tau\right)=1\right)=\\ &=\left|\left(\widetilde{f}_{n,d}+\varepsilon_{n,m,d}\right)-\left(\widetilde{f}_{n,d'}+\varepsilon_{n,m',d'}\right)\right|\leq\tau=\\ &=\begin{cases}P\left(\left|\widetilde{f}_{n,d}-\widetilde{f}_{n,d'}+\varepsilon_{n,m,d}-\varepsilon_{n,m',d'}\right|\leq\tau\right) & d\neq d',\forall n\in[0,N-1],\forall m\in[0,M-1]\\ P\left(\left|\varepsilon_{n,m,d}-\varepsilon_{n,m',d'}\right|\leq\tau\right) & d=d',\forall n\in[0,N-1],\forall m\in[0,M-1]\end{cases}\end{aligned} \tag{4.11}$$

Since involved random variables are characterized by gaussian distribution, which parameters have been illustrated in Tables II and IV, each probability can be easily evaluated by means of the primitive $\operatorname{erf}(x)$. Hence, the Equation 4.11 can be specialized in:

$$\begin{aligned}p &= P\left(\chi\left(\left(\widetilde{f}_{n,d}+\varepsilon_{n,m,d}\right)-\left(\widetilde{f}_{n,d'}+\varepsilon_{n,m',d'}\right),\tau\right)=1\right)=\\ &=\begin{cases}\displaystyle\int_{-\tau}^{\tau}\frac{1}{\sqrt{2\pi\left(2\sigma_{\text{intra}}^2+2\sigma_{\varepsilon}^2\right)}}e^{-\frac{\tau^2}{2\left(2\sigma_{\text{intra}}^2+2\sigma_{\varepsilon}^2\right)}} & d\neq d',\forall n\in[0,N-1],\forall m\in[0,M-1]\\ \displaystyle\int_{-\tau}^{\tau}\frac{1}{\sqrt{2\pi\left(2\sigma_{\varepsilon}^2\right)}}e^{-\frac{\tau^2}{4\sigma_{\varepsilon}^2}} & d=d',\forall n\in[0,N-1],\forall m\in[0,M-1]\end{cases}\end{aligned} \tag{4.12}$$

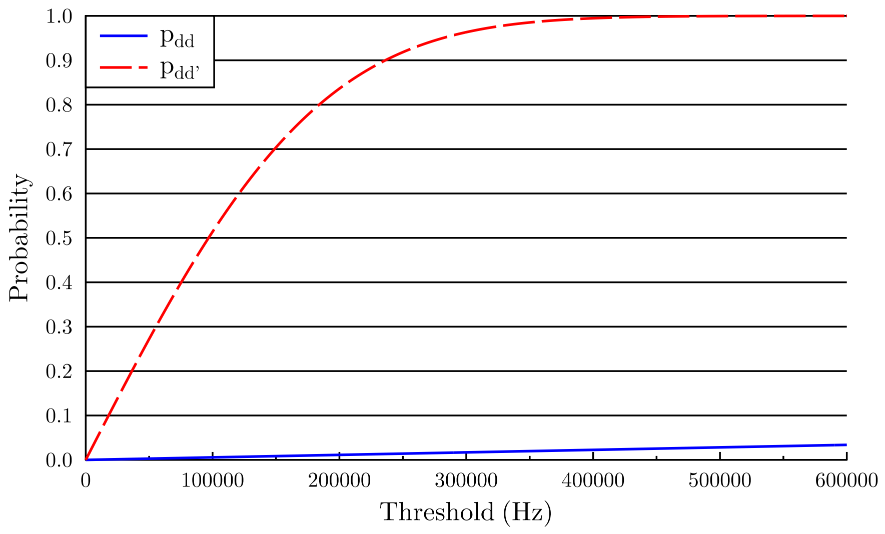


FIGURE 4.19: p_{dd} and $p_{dd'}$ evaluated by varying the threshold τ.

In particular, frequency variables picked from different devices are distributed as $\mathcal{N}\left(\mu, \sigma^2_{\text{intra}}\right)$, hence when $d \neq d'$ the variance associated with $\left(\widetilde{f}_{n,d} + \varepsilon_{n,m,d}\right) - \left(\widetilde{f}_{n,d'} + \varepsilon_{n,m',d'}\right)$ is $\left(2\sigma^2_{\text{intra}} + 2\sigma^2_{\varepsilon}\right)$.

Since the probability p has 2 different forms, which depend on the fact that d and d' are the same device or not, let $p_{dd'}$ be the value given by Equation 4.12 in the first case and p_{dd} the value in the second case. Consequently they can be evaluated as:

$$p_{dd} = \frac{1}{2}\left(1 + \operatorname{erf}\left(\frac{\tau}{\sqrt{2\left(2\sigma^2_{\varepsilon}\right)}}\right)\right) - \frac{1}{2}\left(1 + \operatorname{erf}\left(\frac{-\tau}{\sqrt{2\left(2\sigma^2_{\varepsilon}\right)}}\right)\right) \tag{4.13a}$$

$$p_{dd'} = \frac{1}{2}\left(1 + \operatorname{erf}\left(\frac{\tau}{\sqrt{2\left(2\sigma^2_{\text{intra}} + 2\sigma^2_{\varepsilon}\right)}}\right)\right) - \frac{1}{2}\left(1 + \operatorname{erf}\left(\frac{-\tau}{\sqrt{2\left(2\sigma^2_{\text{intra}} + 2\sigma^2_{\varepsilon}\right)}}\right)\right) \tag{4.13b}$$

The Figure 4.19 reports graphs for both the probability p_{dd} and $p_{dd'}$. In particular, the curve of p_{dd} increases faster than the $p_{dd'}$, since the variance associated with the gaussian distribution for $p_{dd'}$ is greater than the one of p_{dd}, which contains only the variance of two measurements gaussian noise processes. Therefore, this guarantees that it is possible to pick a threshold τ such that $p_{dd} \approx 1$ keeping $p_{dd'}$ still at a very low value near 0.

As for the FAR and the FRR, they can be estimated by exploiting the Equation 4.10. FAR can be expressed as the probability to have a false negative is given by the complementary probability to have at least l χ tests on N trials which succeed given that

$d = d'$:

$$\text{FAR} = 1 - \sum_{i=l}^{N} \binom{N}{i} p_{dd}^{i} \left(1 - p_{dd}\right)^{(N-l)} \tag{4.14}$$

The FRR can be calculated as the probability to have at least l χ tests on N trials which succeed given that $d \neq d'$, hence:

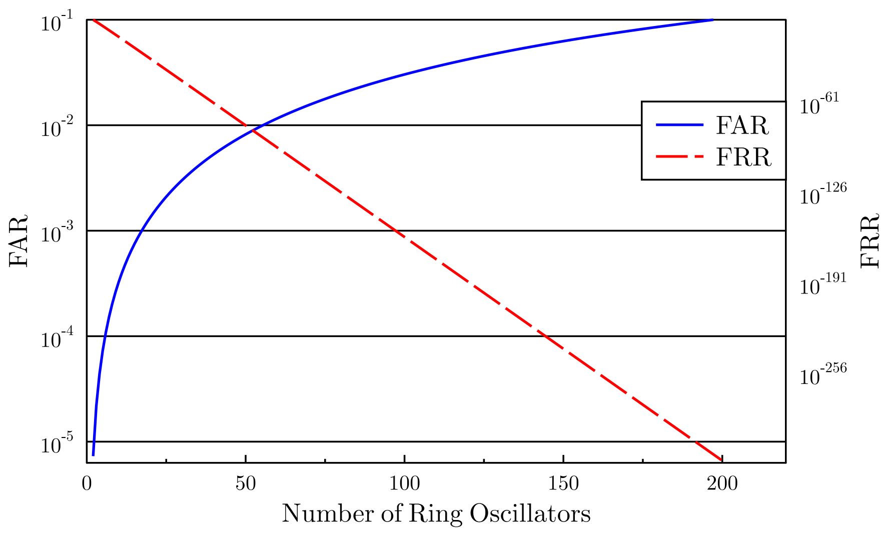
$$\text{FRR} = \sum_{i=l}^{N} \binom{N}{i} p_{dd'}^{i} \left(1 - p_{dd'}\right)^{(N-l)} \tag{4.15}$$

Even if not explicitly reported, the FAR and the FRR depend not only on l and N, but also on τ, σ_ε, σ_{intra}. In particular, the last two parameters are given by the particular implementation of ROs and by the technological target. For the Xilinx Spartan-6, they are reported in Tables II and IV. In Figure 4.20, FAR and FRR are evaluated varying N, τ and l. In particular, the Figure 4.20A shows the two rates varying N, with $l = N - 1$ and $\tau = 3\sqrt{2}\sigma_\varepsilon$. The picked value of the threshold guarantees that 99.7% of the components of $W_{d,d'}$ belongs to the interval $[-\tau, \tau]$ when $d = d'$, i.e. when $W_{d,d'}$ contains only the measurement error. The probability to reject a legit signature grows with the number of involved ROs, since it is less probable to have a positive response with a greater number of χ tests, and the probability to accept a non-legit signature decreases, since it is unlikely that $N - 1$ χ tests give a positive response. Figure 4.20B illustrates FAR and FRR varying the threshold value keeping $N = 9$ and $l = 8$. In particular, the FAR decreases with more permissive threshold values, because it is unlikely that χ tests give a positive response with tight ranges, and the FRR increases since the probability to have a positive response from χ tests grows with the threshold. At the end, the Figure 4.20C shows the impact of the minimum score value l on the rates keeping constant $\tau = 3\sqrt{2}\sigma_\varepsilon$ and $N = 25$. The FAR is 0 up to $l = 18$, then increases up to value near 1, while the FRR decreases. The behavior is similar varying the number of ROs (Figure 4.20A).

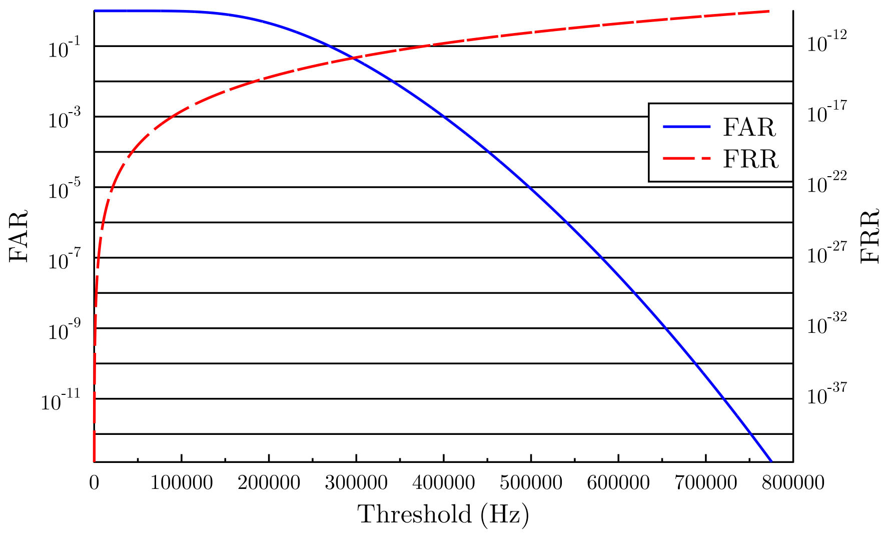
4.3.3.3 Statistical model parameters

The previous Subsection illustrates a statistical model able to evaluate the FAR and FRR values, taking into account 5 parameters:

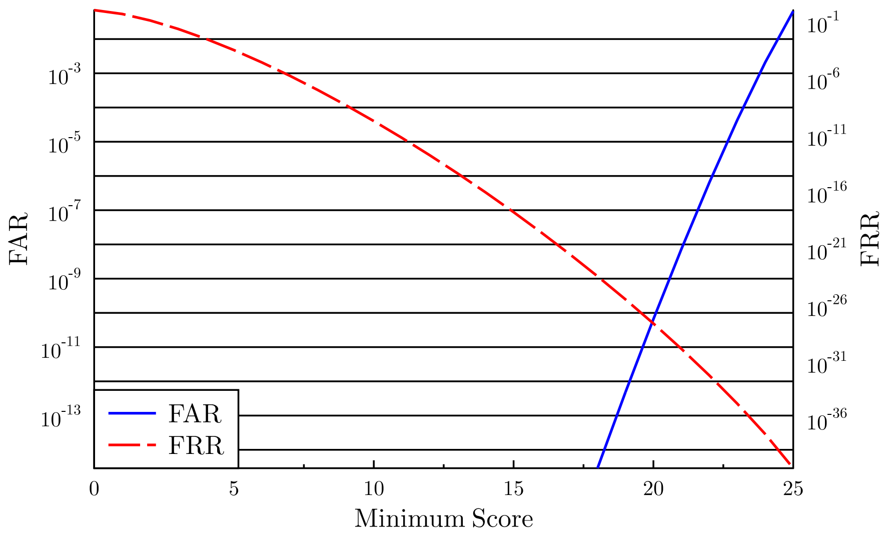
1. σ_ε, the standard deviation associated with the measurement error;



(A) FAR and FRR evaluation varying with the number of involved ROs N.



(B) FAR and FRR evaluation varying with the threshold τ.



(C) FAR and FRR evaluation varying with the minimum score value l.

FIGURE 4.20: FAR and FRR evaluated trough the Equations 4.14 and 4.15 varying the number of ROs, the threshold and the minimum score value.

2. σ_{inter}, the standard deviation associated with the frequency distribution;

3. τ, the threshold for the χ tests;

4. N, the number of involved ROs;

5. l, the minimum number of χ tests required by the function ξ to give a positive result;

As for the first two parameters, they depend only on the technological target in which the FS PUF has to work and on the design of both the measurement architecture and ROs. Indeed, σ_{ε} and σ_{inter}, discussed in Subsections 4.2.4 and 4.3.1, are different changing the number of stages in the loop. Hence, the only way to manipulate them is working with the number of stages, routing strategies, etc. Furthermore, the measurement error can be reduced considering wider counters and slower system clocks.

For the FPGA technology, the estimation of such quantities is not trivial, since the only way to obtain them is just to run some experiments to retrieve as many frequencies as possible and try to estimate them. It is hard to exactly know how many experiments are enough to obtain a confident approximation for both the parameters. For the ASIC technology, the evaluation of such two parameters can be accomplished by means of simulation campaigns (e.g. exploiting the Monte Carlo method), hence there is no need to have available some manufactured ICs.

Consequently, the other parameters can be tuned in order to obtain the wanted FAR and FRR. As indeed, the statistical model provided by Equations 4.14 and 4.15 can be used only to retrieve an estimation of the rates knowing the 5 parameters, but it is not possible to query which parameters can satisfy given values of FAR and FRR. But at least the Figure 4.20 shows the trends of both the rates varying them one by one.

4.3.3.4 Consideration about t_m

The introduced score test relies on the signature $\widetilde{S}_{m,d}$ reported in the Equation 4.9. In particular, $\widetilde{S}_{m,d}$ is defined through an approximation because of the average value of the measurement error. But another source of uncertainty is given by the assumption that $t_{n,m} = t_m, \forall n$. Indeed, even if this approximation introduces a small error, the score test can also handle it in an easy way. In fact, since t_n is distributed as a gaussian (see Figures 4.17 and 4.16), the value $t_{n,m} - \underset{n}{E}[t_{n,m}|m]$ involved in the computation of $\widetilde{S}_{m,d}$ is characterized by a standard variation which can be estimated trough previous experimental campaign and, for the sake of simplicity, it can be included within the σ_ε. Since the contribution of the σ_t is extremely smaller than σ_ε (10^1 Hz against 10^5 Hz), the FAR and FRR modestly benefit.

4.3.4 Experimental result

Exploiting the frequencies value collected during the experimental campaigns illustrated in the Subsection 4.2.4, the score test can be evaluated on a significant amount of real data; in particular three data sets can be analyzed:

- normal working conditions: 10 devices, 938 ROs per device, 25 measurements for each RO;
- temperature variations: 1 device, 938 ROs, 9 measurements for each RO, 6 temperature values;
- aging: 1 device, 938 ROs, 20 measurements for each RO, 2 characterization (fresh and aged).

The results in terms of FAR and FRR are reported in Table V. They were calculated by tuning the l value, i.e. the minimum score which establish if two signatures are produced from the same device or not (see ξ function in the Equation 4.10). In the case of uncontrolled working conditions, i.e. without keeping the temperature stable, even with only 2 ROs the FAR is 0 and FRR is 9.239e-03 and with 5 ROs. It is worth noting that with only 2 ROs the RO PUF is able to discriminate only 2 device since it can produce only one response bit. Contrary to this case, the FS PUF is discriminating $\frac{10\times938}{9}$ devices. As for the temperature case, the best conditions are with 9 and 10 ROs, while for the aging the best conditions are with 7 and 8 ROs.

Number	Uncontrolled Conditions		Temperature Variations		Aging	
of ROs	FAR	FRR	FAR	FRR	FAR	FRR
2	0	9.239e-03	3.217e-08	8.152e-02	2.011e-09	7.731e-02
3	2.001e-09	3.703e-02	1.044e-08	1.703e-02	1.233e-08	3.099e-02
4	0	1.268e-03	7.150e-09	3.484e-02	6.255e-10	5.211e-02
5	0	0	0	1.103e-02	0	2.893e-03
6	0	0	0	0	4.924e-08	3.782e-03
7	0	0	5.267e-07	1.137e-04	0	0
8	1.739e-11	6.641e-07	0	2.259e-06	0	0
9	0	0	0	0	2.152e-10	0
10	0	0	0	0	0	2.439e-09
11	0	0	0	2.747e-13	0	0

Table V: FS PUF evaluated in terms of FAR and FRR trough experimental campaigns conducted on the Xilinx Spartan-6 FPGA family. The value are obtained by tuning the minimum value of the score.

Chapter 5

Network Traffic Analysis: a Case Study

This Chapter illustrates an exhaustive case study with a twofold aim:

- demonstrate that the FPGA is suitable to implement high throughput algorithms, accelerating their execution with respect to a software implementation;
- show the feasibility of the deploying an application in a large-scale scenario, involving hardware in-field updates.

In order to match both the goals, the case study focuses on a network traffic analysis hardware accelerator distributed among a smartphones population. The network analyzer is based on Decision Tree (DT) predictor algorithm and it is implemented as hardware accelerator of a mobile node, which hardware technology is based on FPGA. Moreover, the generation of such accelerator is made automatically through a methodology that involves a software algorithm, which transforms predictor model based on DT into HDL projects, and a distributed infrastructure, which distributes new hardware version of DT and collects new threats from the device population.

5.1 Traffic Analysis for Security Purposes

Mobile traffic is significantly growing, thanks to the increased access capacity provided by 3G and 4G technologies and to the rising computing power of the latest smart devices. According to Cisco's Global Mobile Data Traffic Forecast Update [88], global mobile data traffic grew 81% in 2013, reaching 1.5 exabytes per month at the end of the year. Most of this traffic is generated by smart devices including smartphones, tablets and wearable devices, which are increasingly adopted in domains where sensitive information are exchanged and processed, such as entertainment, healthcare and business applications. Traffic monitoring and traffic analysis are fundamental means to acquire information about incoming flow for the detection of anomalous behaviors, viral infections and hacking attempts, in addition to being a powerful tool to define accurate network models, validate new protocols and applications, diagnose network failures, and ultimately enhance network performance and quality of service. The application of traffic analysis techniques to the mobile domain is not straightforward, mainly due to the devices' distributed nature and resource constraints. Because of the large number of devices and the huge amount of involved traffic data, performing traffic analysis as a centralized task would sensibly reduce network performance and may not be a viable solution. On the other hand, running complex traffic analysis tools on a mobile device would heavily affect the device's energy consumption.

Moreover, due to applications' dynamicity, traffic models used by mobile devices would rapidly become outdated, thus requiring periodic updating to include new threats and to possibly take into account the feedback coming from other devices. Machine learning algorithms have been widely adopted for traffic analysis, thanks to their flexibility and the capability of dynamically generating updated models from feedback knowledge. These techniques, which typically require high computational capabilities for large data sets, can be successfully implemented in hardware, by adopting, for example, hardware accelerators, which allow complex computations at reduced power. However, even when providing the mobile nodes with the capability of processing traffic data by means of advanced algorithms implemented in hardware, the efficacy of the traffic analysis mechanism is still limited, due to the fact that the traffic model is statically embedded in the hardware configuration. Indeed, the recent advances in hardware reconfiguration technology would allow to dynamically reconfigure the hardware design of a physical device, thus enabling the dynamic update of the traffic model according to network behavior, in order to enhance classification accuracy of network packets.

With respect to the accuracy of traffic models used by devices, it is quite evident that the traffic involving a single device may not be enough to carry out complex evaluations, while the correlation of traffic data involving heterogeneous and geographically distributed sources may sensitively enhance the model used for traffic classification. In such scenario, of course, a direct exchange of information among nodes is not feasible and a different communication paradigm is required. In particular, some sort of centralized or distributed processing entity must be envisioned, provided with specific features, i.e.: it must be easily accessible from mobile nodes, physically located anywhere and represented by heterogeneous devices, and it must allow such nodes to share information about unclassified traffic flows or detected threats and to obtain a security characterization of unclassified suspicious traffic. The service-oriented paradigm is particularly suited to cope with the above-mentioned requirements: services can be easily invoked through simple APIs by heterogeneous devices, and are being increasingly adopted in mobile applications thanks to their flexibility.

5.2 Related Research Efforts

Recently, in the literature, a significant number of research papers on mobile traffic analysis have arisen, with a special attention to the smartphone domain, trying to characterized the involved traffic and providing some architectural solutions for the security. Authors of [40] presented a detailed analysis of smartphones' traffic by collecting traffic data from users' terminals. In [86], the authors proposed an innovative approach to develop a honeypot with aim to collect malwares in FPGA architecture, they also used partial reconfiguration to update the network interface to reject the new threats found.

As for security issues, since the smartphones malware are significantly growing in number, several techniques, mainly based on intrusion detection features, have been proposed to protect mobile devices. Luo et al. [69] presented a security framework to protect smartphones, listing security issues and the techniques, by means of architectural modules, that are able to deal with each of them. Even if an intrusion detection system (IDS) for network activities analysis against policy violations was designed, they did not discuss neither an implementation nor the efficacy of this module. In an analogous manner, the authors of [97] devised an IDS specifically tailored for smartphones, named SIDS, but even in this case, neither details were given about the implementation, supposed to be in software, nor on the resources impact for a mobile system. More recently, the authors of [10] integrated in Android a network classifier, based on DT, executing a software routines on incoming packets demonstrating the efficacy of the approach. They illustrated details on involved features and algorithms, but no performance details on a real system were given. Unlike [10], the aim of this case study is to implement the same approach, but exploiting a real reconfigurable hardware device rather than a software approach to accomplish the traffic analysis.

As for the classification architecture, several FPGA-based traffic analyzer implementations have been proposed in order to speed up the classification processes [99, 101]. The most suitable architecture is the one based on DTs, since their adoption does not require arithmetic operations, which are expansive to implement in hardware, but only comparisons, easy to realize. In [114] two architectures were proposed to classify packets traffic using FPGAs. Both the architectures use a programmable classifier exploiting the C4.5 algorithm. Using NetFPGA, in [82] the authors proposed a traffic classifier by using C4.5. The main architectural characteristic is the programmability by the software, without loss of service, using memories that store the classifier. The C4.5 classification algorithm is very promising for big data analysis, it performs very well in many application domains as the predictor works on the tree structure built during the learning phase. It was presented by Quinlan in [93] and soon was adopted in a broad range of applications such as image recognition, medical diagnosis, fraud detection and target marketing.

In [43] and [81] the C4.5 algorithm was adapted within the framework of differential privacy in distributed environments. In [67] the authors showed the power of C4.5 for classifying the Internet application traffic, due to the discretization of input features done by the algorithm during classification operations. The authors of [117] extended the traditional decision tree based classifiers to work with uncertain data. The results presented in these papers are very interesting, but they did not provide any automatic tool to automatically generate the hardware architecture from the prediction model.

Algorithm 1 Binary Decision Tree predictor algorithm.

Require: A feature vector f_{vector} and a classification model
Ensure: The predicted Class for the input f_{vector}

```
node ← model.root
while !node.isLeaf do
    if node.$\rho\left(f_{vector}\left[\text{node.f}\right], \text{node.k}\right)$ then
        node ← node.childLeft
    else
        node ← node.childRight
    end if
end while
return node.classValue
```

5.3 Decision Tree Hardware Implementation

Among available machine learning approaches, the literature proved the DT machine learning approach to be one of the most effective in traffic analysis, since it is able to achieve high efficacy in discriminating traffic behavior. DTs are tree-based predictor models which consist in internal nodes, containing rules, and leaves, labeled with a classification value: basically, a DT represents a set of rules which classify data according to the conditions specified by the nodes. Generally a condition specifies an attribute (feature) in the dataset, a relational operator and a constant. A condition may contain a more complex expression over features, but here this case is not considering. Let $D_{\rho}^{k}\left(f\right)$ be a decision node which requires the feature input f and configured with parameters ρ and k. In particular, $D_{\rho}^{k}\left(f\right) = \rho\left(f, k\right)$, $\rho \in \{<, \leq, >, \geq, =, \neq\}$ and k a constant involved in the comparison. For instance, a decision could regard the TCP/UDP port number: $D_{\leq}^{1024}\left(\text{port}\right) \leq \left(\text{port}, 1024\right) = \text{port} \leq 1024$. Each decision returns a boolean value, since the result is ever expressed with positive or negative response.

As for the predictor algorithm, it has to visit the tree according to the conditions of each nodes, as specified by the Algorithm 1: starting from the root of the tree, the evaluation of such conditions determines which child node has to be selected for the next evaluations. At the end, when a leaf has been reached, the label of the leaf is returned as the decision value of classification. Without loss of generality, the trees considered here are only binary: indeed it is always possible to manipulate a n-ary tree to obtain a binary one. A prediction algorithm is able to visit a binary DT by evaluating the condition in each node and it steps to left if the condition turns out true, otherwise to right. In Figure 5.1 there is an example of a binary Decision Tree: it is made of

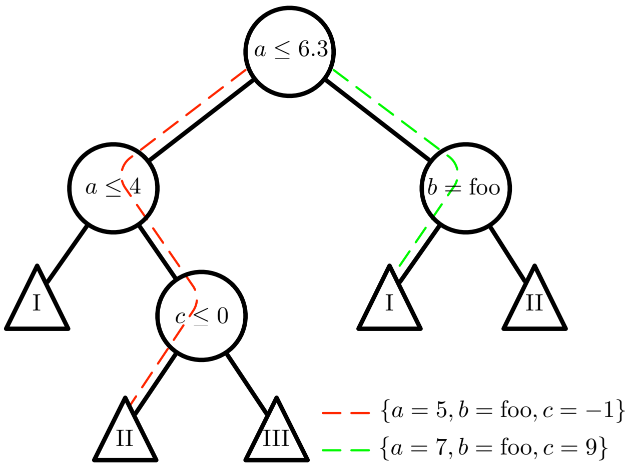

$a \leq 6.3$
$a \leq 4$
$b \neq \text{foo}$
$c \leq 0$
I
I
II
II
III
$\{a = 5, b = \text{foo}, c = -1\}$
$\{a = 7, b = \text{foo}, c = 9\}$

FIGURE 5.1: An example of Decision Tree with 4 nodes, 3 features and 3 classes. The colored dotted lines indicate the paths for the two features values examples.

4 nodes evaluating conditions on 3 features and its leaves are labeled with 3 different classes. The dotted lines represent the path of two example inputs: the red is the path for $\{a = 5, b = \text{foo}, c = -1\}$, the green for $\{a = 7, b = \text{foo}, c = 9\}$.

5.3.1 Programmable decision trees

The non-programmability characteristic of the hardware makes the DT accelerator implementation difficult to realize. As said before, DT predictor models are continuously updated, especially in the network domain. In the literature, some programmable DT accelerator architectures are available. They are designed in the structure, but their behavior is determined through a configuration. The advantage, of course, is the programming at runtime, but they are memory-based, resulting slower or bigger in the area than fixed implementations. Also, since they have to be realized once, resulting accelerators are static in parameters, meaning that such parameters cannot be changed. For instance, in [16] the tree visiting accelerator unit can be programmed to compute the prediction algorithm on different trees, but it is designed to execute the algorithm over a maximum number of nodes. Hence, if an application requires a predictor which nodes amount is bigger than the maximum available on the accelerator, the unit cannot be used.

The SRAM based FPGA can mitigate the problem because it provides the DPR. Hence, a hardware accelerator can be designed on a FPGA, without providing additional memory banks for the core configuration, and allowing the updating of it during the time. Of course, the core update problem is not completely solved since the FPGA could potentially not have enough available resources (such as clock distribution circuitry, look-up tables, memory banks) in the dynamic partition to host future versions of the accelerator that could be bigger in area.

5.3.2 Implementing static Decision Trees

As shown in the Algorithm 1, the predictor procedure is strictly sequential and cannot be parallelized. In fact, each cycle decides, trough the evaluation over a feature value, for the next cycle that has to be take into account, and so on. To reach high throughput in the DT visiting avoiding sequential computations, a speculative approach can be considered. The main idea is to evaluate which leaf is reached by computing in parallel all the nodes conditions. The latter is performed by fast comparators, named Decison Boxes, which work in parallel. Then, a boolean net decides, evaluating the outputs of the Decision Boxes, which leaf, hence which class, the algorithm returns. In the evaluation performed by the hardware accelerator, the speculation stands in computing some conditions of the tree that are useless for the algorithm result, such that the Algorithm 1 would never evaluated.

To better understand, consider the Figure 5.2 which depicts the hardware accelerator unit for the DT prediction give in Figure 5.1. As one can see, considering the input $\{a = 5, b = \text{foo}, c = -1\}$ (the same input reported in Figure 5.1), the result is independent from the evaluation performed by the Decision Box over b. Indeed, the node with condition $b = \text{foo}$ does not belong to the path that the visiting algorithm generates with the input that is considering (Figure 5.1, red dotted line).

Boolean net details The boolean net is composed by different boolean functions in form of *sum of product* (SoP), as many classes as in the predictor model. Let BF_C the SoP form for the class C. This function has to rise logic-1 when at least one of the paths that leads to a leaf that is labeled as C is activated. Hence, to define the boolean function for a given class C it is necessary to collect all the paths that lead to leaves

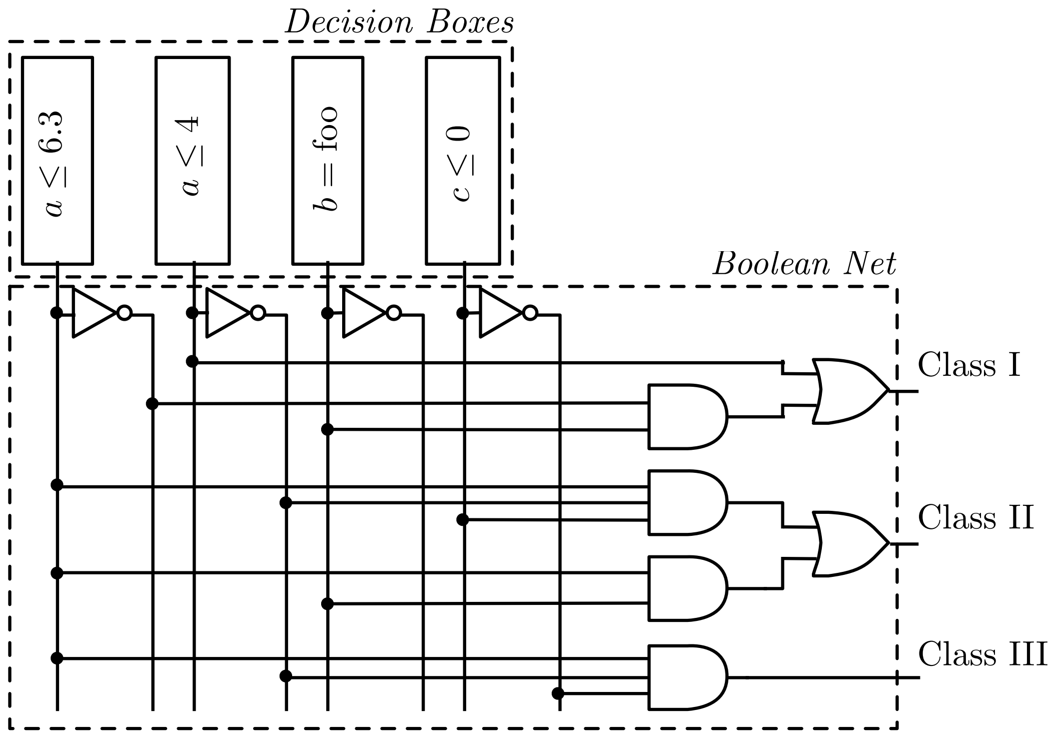


FIGURE 5.2: Hardware implementation of the prediction algorithm for the tree in Figure 5.1

of C. Let $P_C(i)$ be the i-th path that leads to the i-th leaf which is labeled as C. Let the assertion $A_C(i)$, $i \in \{0,\ 1,\ ...,\ leavesOf(C) - 1\}$ be the logical intersection of all decisions met along $P_C(i)$. The decisions are literals of this boolean function and can be taken direct or negated. In other words, an assertion $A_C(i)$ is the intersection of all intervals, defined by decisions $D_\rho^k(f) \in P_C(i)$, in which the input features belong to such that they are representative of the class C. Formally, each assertion can be defined as:

$$A_C(i) = \prod_{D_\rho^k(f) \in P_i} D_\rho^k(f).$$

Finally, for each class C it is possible to define the boolean function as:

$$BF_C = \sum_{i=0}^{leavesOf(C)-1} A_C(i) = \sum_{i=0}^{leavesOf(C)-1} \prod_{D_\rho^k(f) \in P_C(i)} D_\rho^k(f).$$

An important observation to do on C4.5 algorithm is that some assertions may be redundant. In fact, considering the example given in Figure 5.1, the paths to reach leaves labeled with the class I, are: $P_I(0) = \left\{D_{\leq}^{6.3}(a), D_{\leq}^{4}(a)\right\}$ and $P_I(1) = \left\{\overline{D_{\leq}^{6.3}(a)}, D_{=}^{foo}(b)\right\}$. In particular, $P_I(0)$ has the associated assertion: $A_I(0) = D_{\leq}^{6.3}(a) \cdot D_{\leq}^{4}(a) = D_{\leq}^{4}(a)$. Indeed, if $a \leq 4$ is true, surely also $a \leq 6.3$ turns out true.

Thanks to this observation, the number of literals in each assertion can be reduced. Assertions which exploit this rule are defined as essential assertions. Furthermore, considering only essential assertions, it is possible to found an upper limit to the number of decision boxes needed to evaluate them, that is an important result for the scalability of the architecture.

Theorem 5.1. *Essential assertion $A_C(i)$ contains at most $2 \cdot |F|$ literals.*

Proof. Let us consider the feature set $\{F\}$ and $D_{\rho}^{k}(f) \in P_C(i)$.

Degeneracy Case:

$f \in \{F\}$ does not appear in any $D_{\rho}^{k}(f) \in P_C(i)$, this means that it has not be chosen by the learner as discriminating feature for the class C.

Case 1:

$f \in \{F\}$ is a continuos feature that appears at least once. Thus $\rho \in \{<, \leq, >, \geq\}$, $f \in \mathbb{R}$. $D_{\rho}^{k}(f)$ decides an interval $\forall f$:

- right interval of k ($[k, +\infty[$ or $]k, +\infty[$), if $\rho \in \{>, \geq\}$;
- left interval of k ($]-\infty, k]$ or $]-\infty, k[$), if $\rho \in \{<, \leq\}$.

If in $P_C(i)$ the decision appears complemented, it is possible to simply choose the complemented interval.

If f appears twice or more, $D_{\rho}^{k}(f), D_{\lambda}^{j}(f) \in P_C(i) \Rightarrow D_{\rho}^{k}(f) \cap D_{\lambda}^{j}(f) \neq \emptyset$, because the learner generates the conditions by splitting operation over them, consequently $k \neq j$. So the intersection $D_{\rho}^{k}(f) \cap D_{j}^{\lambda}(f)$ can generate:

1. a right interval if both the decisions define a right interval: $[M, +\infty[$ or $]M, +\infty[$ with $M = max(k,\ j)$,

2. a left interval if both the decisions define a left interval: $]-\infty, m]$ or $]-\infty, m[$ with $m = min\,(k,\ j)$,

3. an interval $[m, M[$ or $]m, M[$ or $]m, M]$ or $[m, M]$, with $m = min\,(k,\ j)$ and $M = max\,(k,\ j)$.

This process can be iterated as many times as f appears in decisions along $P_C(i)$. In the first two cases $A_C(i)$ requires only one decision (literal) on f, in the third an intersection between two decisions.

Case 2:
f is a nominal feature $\Rightarrow \rho \in \{==,\ !=\}$, f can assume finite values. A $D^k_\rho(f)$ appears at most once on any path in the tree. So $A_C(i)$ requires only one decision on f. $\square$

FPGA implementation As previously said, each BF_C is expressed as SoP. Considering the worst case, BF_C has an and gate fan-in equal to $2 \cdot |F|$, and an or gate fan-in equal to $|C| - 1$. The gates implementation delay grows with the size of the fan-in like a step function on the FPGA technology, because the FPGA is able to implement each function in a k-input LUT, consequently to realize gates with higher fan-in it combines more that one LUT in a tree scheme. This scheme, where each gate has a maximum fan-in of k (k-gate), can be successfully exploited to define a pipelined structure in which each LUT output is registered in a dedicated flip-flop.

As for the area occupation, being N the total fan-in an integer multiple of k, and S the depth of the tree, m_i (the number of k-gates at i-th level) is: $m_0 = \lceil \frac{N}{k} \rceil$; $m_1 = \lceil \frac{m_0}{k} \rceil$; $m_i = \lceil \frac{m_{i-1}}{k} \rceil = \lceil \frac{m_{i-2}}{k^2} \rceil = \lceil \frac{N}{k^{i+1}} \rceil$. In the last stage of the pipe there is only one k-gate: $m_{s-1} = 1 = \frac{N}{k^S}$. So the relation between the tree depth S, the original fan-in N and the fixed fan-in k is: $k^S = N$, so $S = \frac{lg(N)}{lg(k)}$, hence the number of k-gates is:

$$\begin{aligned}\#k - gates &= \sum_{i=0}^{S-1} m_i \simeq N \cdot \sum_{i=0}^{S-1} \frac{1}{k^{i+1}} = N \cdot \left(\sum_{i=0}^{S} \frac{1}{k^i} - 1 \right) \\ &= N \cdot \left(\frac{1 - \left(\frac{1}{k}\right)^{S+1}}{1 - \frac{1}{k}} - 1 \right) = \frac{N-1}{k-1}, k > 1.\end{aligned}$$

The total number of gates N_g for all BF_C can be defined as:

$$N_g = \sum_{\forall C} \left(\frac{2 \cdot |F| - 1}{k-1} \cdot leavesOf(C) \right) + \sum_{\forall C} \left(\frac{leavesOf(C) - 1}{k-1} \right). \qquad (5.1)$$

where the first term is related to number of k-gates to implement the and with the worst fan-in ($N = 2 \cdot |F|$), while the second term is related to the or that has a fan-in equal to the number of leaves ($N = leavesOf(C)$). To evaluate the total gates number of the boolean net, two different cases can be considered:

Balanced Case: the leaves are equally distributed to each class; so $leavesOf(C) = \frac{DN+1}{|C|}$. Substituting in 5.1:

$$N_g = |C| \cdot \left(\frac{2 \cdot |F| - 1}{k-1} \cdot \frac{DN+1}{|C|} + \frac{\frac{DN+1}{|C|} - 1}{k-1} \right) = \frac{2 \cdot |F| \cdot (DN+1) - |C|}{k-1}. \qquad (5.2)$$

Unbalanced Case: each class has only one leaf, except for $\hat{C}$ class with $leavesOf(\hat{C}) = DN + 1 - (|C| - 1)$. Substituting in 5.1:

$$\begin{aligned} N_g &= \frac{2 \cdot |F| - 1}{k-1} \cdot (DN - |C| + 2) + \frac{2 \cdot |F| - 1}{k-1} \cdot (|C| - 1) + \frac{(DN - |C| + 2) - 1}{k-1} \\ &= \frac{2 \cdot |F| \cdot (DN+1) - |C|}{k-1}. \end{aligned} \qquad (5.3)$$

In both the equations 5.2 and 5.3 there is a linear dependence of N_g with the number of decision nodes DN, this result demonstrates the scalability of the solution. As for the tree depth, even though the number of gates does not change with the classificator characteristic, S is different and, consequently, the pipe latency too.

5.3.3 Automatic hardware core generation

With the transformation rules given by the Theorem 5.1, an automatic process can be defined such that from the training set it synthesizes the decision tree hardware. The process flow is reported in Figure 5.3; it is made of three different steps that produce in output three standard artifacts. The goal of the Data Processing is to automatically structure data coming from heterogeneous sources into a common schema, hence a data-preprocessing tool extracts the only relevant information to build a common

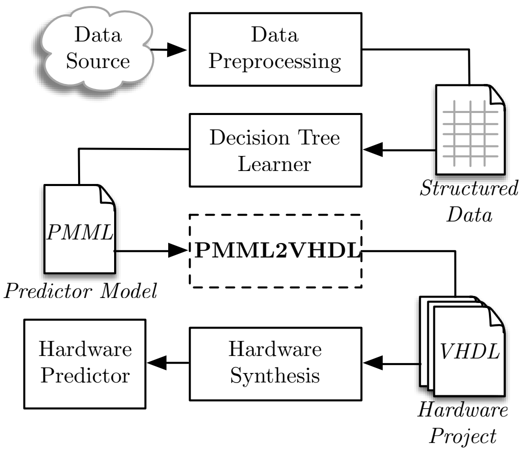


FIGURE 5.3: Automatic process flow for implementing a hardware predictor.

schema. This artifact, stored in a tabular format, is given in input to the model building step. It implements C4.5 learning algorithm, the output of this step is the predictor model coded into PMML (Predictor Model Markup Language), a standard XML schema for predictors. At this step, exploiting the PMML formalization, I developed a tool, PMML2VHDL, that parses the decision tree predictor model and generates an optimized hardware description for the predictor [6]. The tool adopts the VHDL as output language, but other descriptions can be supported in future. Finally, in the last step, the *Hardware Synthesis*, the VHDL is used as input for the hardware synthesizer in order to obtain a working version of the predictor.

For the purposes of this experimental case study, the proposed predictor algorithm is exploited to analyze the traffic, although it can be adopted to implement any DT model, guaranteeing high throughput and feasibility in the implementation [5].

5.3.4 Integration within real device

In order to provide a prototype of device which classifies the Internet traffic with a hardware accelerator, the same Zedroid project exploited in Section 3.6 has been adopted.

Figure 5.4 schematically details the integration of the hardware accelerator for the traffic analysis within the Xilinx Zynq Architecture. As one can notice, the hardware accelerator unit is connected with the processing system by means of two AXI interfaces. They provide the packets received by the Ethernet interface. The configuration uses the

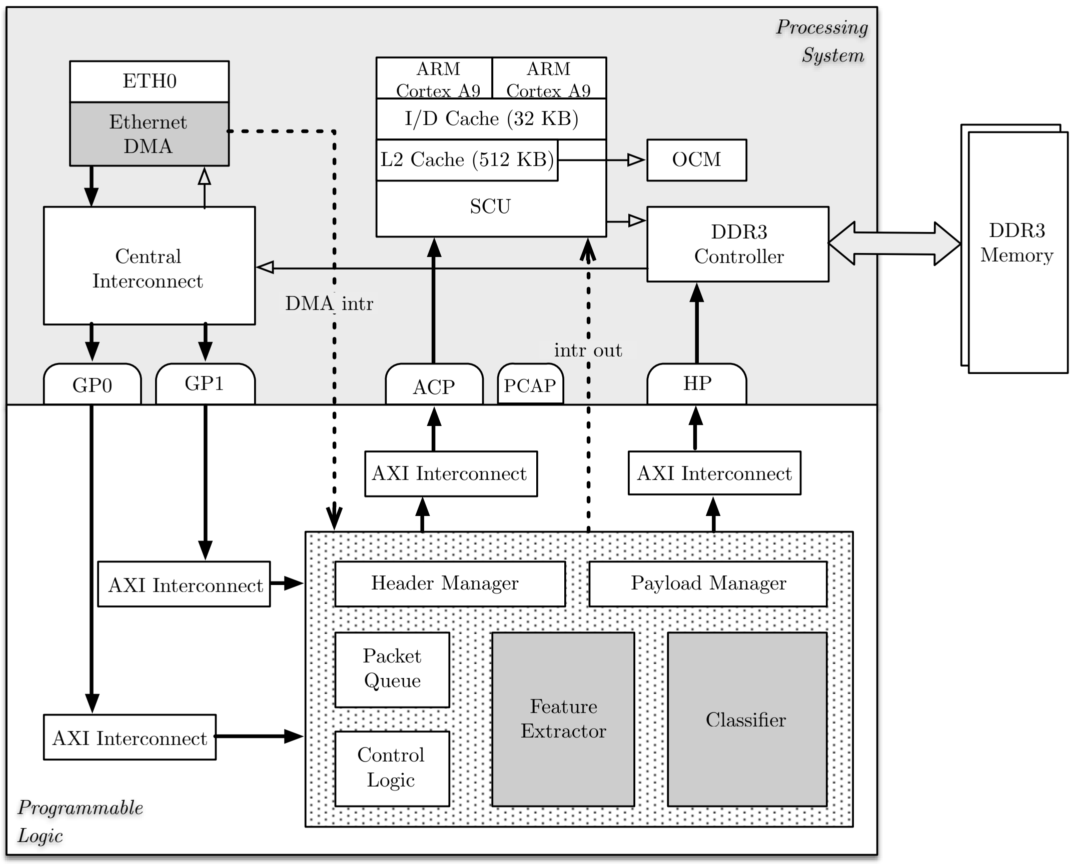


FIGURE 5.4: Hardware overview of the Zynq platform, which is configured with a network analyzer accelerator.

Ethernet wired interconnection, as Zynq does not provide a wireless one. Note that from the analysis point of view there are no differences between these kinds of communication since the analysis does not take into account the packet content.

Normally, the Ethernet DMA copies the received packets directly to the system memory. Conversely, in this specific case, the DMA is configured at system boot to redirect all traffic packets to the Programmable Logic (PL) section, where they are analyzed by a hardware accelerator. As for the predictor hardware accelerator unit, the main block is the Classifier, that detects malicious packets by using their features, properly extracted by the Feature Extractor component. In order to reduce the latency of the approach, each packet is split up into header and payload. Indeed, the custom peripheral separates headers from payloads, copying the former into the level-2 (L2) cache and the latter into the main memory: this way, packets headers will always be available in the high speed L2 cache for the Linux specific tasks [121].

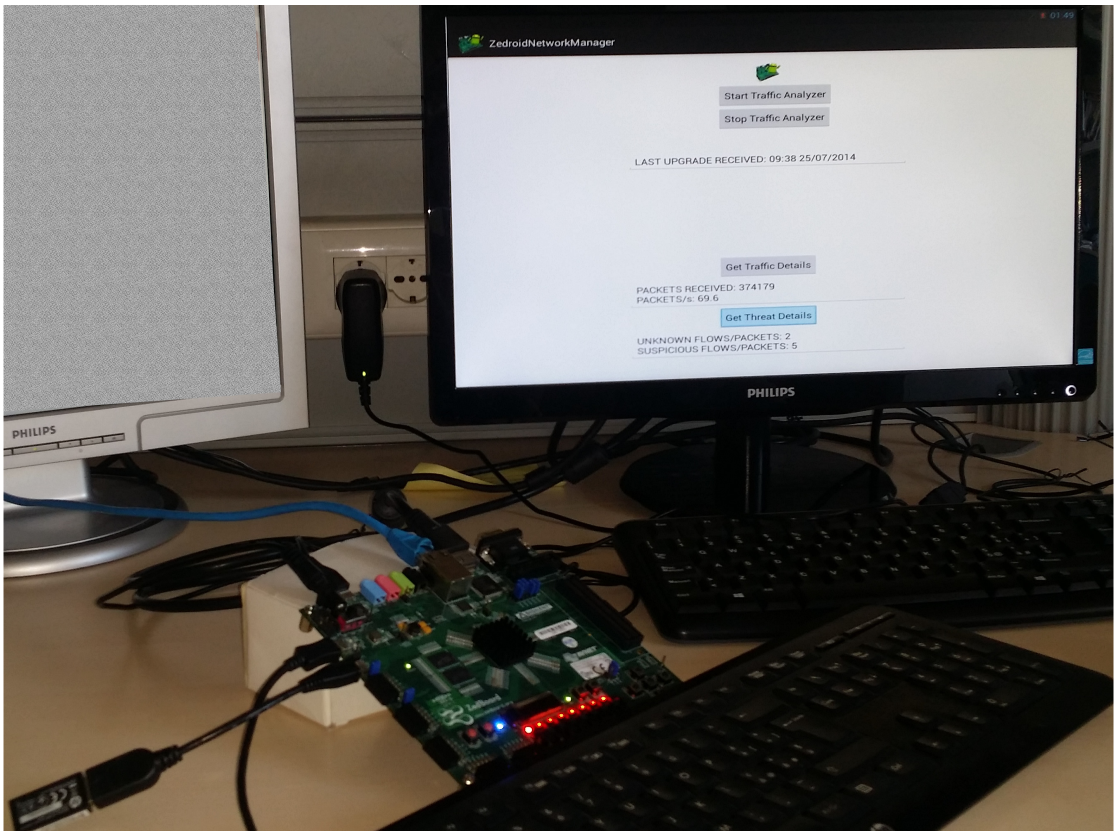
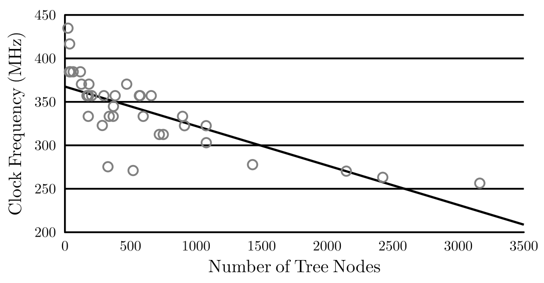


Figure 5.5: Zedroid project is running a simple App to manage the hardware network classifier.

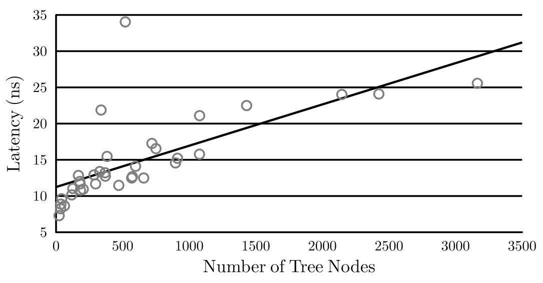
The model extraction has involved the C4.5 predictor algorithm and a data set composed of 10200 packets, which were pre-labeled. Running different prediction algorithm executions, changing each time the data set characteristics, such as the feature considered (from 6 to 20), the classification coarseness (from the binary classifier to 6 different classes), 36 models were collected. Each execution was saved in PMML, converted in VHDL, by means of the tool PMML2VHDL and synthesized by using Xilinx Vivado 2013.4 tool. From each experiment, it is possible to collect: the number of the tree nodes, the accuracy obtained from the model and the dynamic power consumption. The last parameters were retrieved through the following steps:

1. Synthesis and implementation of the core using Vivado tool: during this phase the tool gives the maximum clock frequency by performing a timing analysis, in order to check that both setup and hold timing constrains are not violated.

2. Definition of a test bench for the core: in this phase the test bench is exploited to obtain the Switching Activity Interchange format (SAIF) file, which contains the



(A) Linear interpolation of maximum clock frequency varying on trees nodes.



(B) Linear interpolation of packets latency introduced by the accelerator varying on trees nodes.

FIGURE 5.6: Time analysis of 30 experiments conducted on the Zynq FPGA architecture.

switching activity for the core under test;

3. Power analysis: exploiting the SAIF, environmental parameters (junction temperature, ambient temperature, airflow, board temperature, etc.), voltage supplier values and the working frequency, it is possible to evaluate both static and dynamic power consumption.

5.3.5 Time performance

To evaluate the latency introduced by such traffic analysis, performed by the hardware accelerator, it is necessary to firstly measure the the clock frequency varying the number of the nodes. In Figure 5.6A the maximum clock frequency varying on the number of nodes are fitted with a linear function. The clock frequency is related to the worst logic delay in the circuit. As said before, the implemented logic is realized in pipeline fashion. So with the tree growing, the FPGA routing introduced delay between the pipeline stages, that has a linear trend. The result implies that, also with big tree structure, the hardware accelerator can work at high clock frequency values that varies linearly with the tree nodes.

In Figure 5.6B, the latency linearly grows with the growing of nodes. This happens because the pipeline stages, and their delays, increase with the number of decision boxes. As the latency is the time value between the feature inputs and their classification output, the result indicates that the time to have the classification result is very short also with big tree structure.

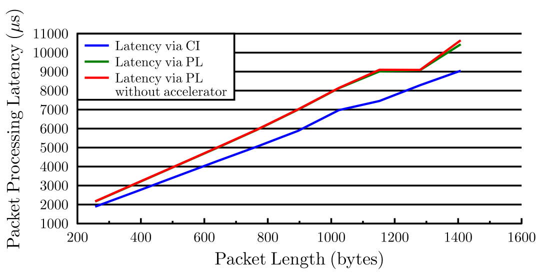


FIGURE 5.7: Global packets latency, measured using the Central Interconnection and redirecting the traffic on the programmable logic through loop-back traffic generation.

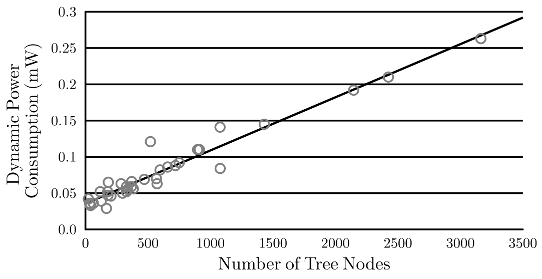


FIGURE 5.8: Dynamic power consumption against the number of trees nodes.

Loop-back measurement In order to measure the time to complete a loop, so the transmission and the receipt a packet, the most effective model was picked, i.e. the one with the highest accuracy in classification, and was integrated in the system. Some traffic flows were produced by an Android app; they differed from one another in the size such that it was possible to appreciate the global delay in payload traversal through the programmable logic, with and without the traffic analyzer, into the central memory. Figure 5.7 shows the loop-back experimental result. The traffic redirection causes a latency increment, due to the overhead introduced by the bus infrastructure on the PL of the FPGA and the latency introduced by the hardware accelerating is very small.

5.3.6 Energy performance

As for the energy consumption, Figure 5.8 illustrates the dynamic power varying on the number of nodes. There is a linear dependency of the energy consumption with the number of nodes. As for the software implementation of the same algorithm, the computational complexity of tree visiting is $O(log_2(n))$, with n the number of tree nodes. This means that the time and the energy consumption for the software implementation have a logarithmic trend, but for the discussed approach they have a linear behavior, even if the accelerator architecture exploits a speculative technique. However, comparing software and hardware approaches, a huge difference in terms of throughput can be noticed [5].

5.4 Distribution Infrastructure

As anticipated before, in Section 5.1, the service-oriented paradigm is particularly suited to cope with the above-mentioned requirements: services can be easily invoked through simple APIs by heterogeneous devices, and are being increasingly adopted in mobile applications thanks to their flexibility. For this purposes, a two-tier service-based traffic analysis infrastructure can be envisioned. Figure 5.9 proposes a service architecture. In particular, at the mobile network layer, nodes run a high-accuracy hardware traffic analyzer based on the embedded traffic model. On occurrence of specific events (e.g., inability in classifying a packet or a packet flow), a node may invoke the service layer through proper APIs, in order to get information about unclassified traffic and to receive updates of the model, represented by a partial bitstream to re-configure the node's hardware with. At the service layer, traffic data involving heterogeneous and geographically distributed sources are correlated and processed with respect to the latest information about security threats and application vulnerabilities, to produce enhanced traffic models, which can be distributed to devices in an on-demand fashion according to the as-a-Service paradigm.

The service layer must be provided with significant computational power in order to be able to update the traffic model according to node-generated information. At the same time, it must be equipped with large and scalable storage capability, in order to cope with intensive and dynamic traffic flows coming from multiple sources. All the above discussed requirements represent a technical challenge which can be effectively solved by means of the adoption of the cloud computing paradigm: it features flexible high-performance computing and high extensible data storage, and therefore it is particularly suited to dynamically process huge amount of distributed data, in order to perform network traffic analysis and target packet dissection for misclassified packets.

As anticipated, the service layer can be designed as a cloud computing infrastructure, able to cope with flexibility, scalability and high-computation and storage requirements typical of a large-scale traffic analysis application. The way in which the cloud-based traffic analysis application is offered to devices in the given context is well represented by the Software-as-a-Service (SaaS) model. From the business point of view, in the illustrated architecture, a cloud provider offers a traffic analyzer software to its customers,

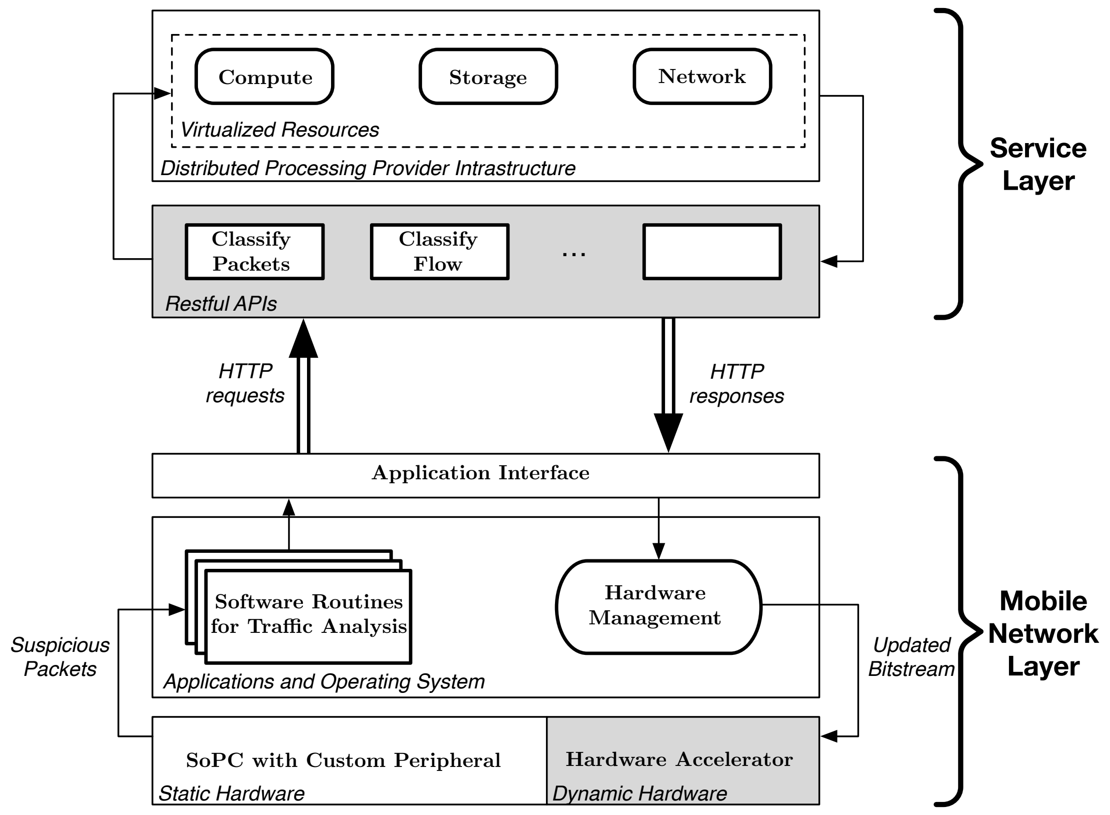


FIGURE 5.9: Overview of a two-tier service-based traffic analysis infrastructure.

represented by the mobile devices owners, on a per-user/pay-as-you go basis, applying the scheme proposed in the Chapter 3.

As an example, the cloud provider may allow all registered (free) customers to download a basic configuration for the hardware accelerator implementing the local traffic analyzer and to obtain periodic updates for it, taking advantage in turn of the amount of precious data coming from the devices needed to enhance the model. On the other hand, premium customers may be provided with more sophisticated models and with more frequent updates.

Several technologies may be adopted to implement the mentioned cloud services. As here the main focus is in on the whole architecture and not in a particular implementation of each service, without loss of generality it is legit to assume that the cloud provider offers RESTful based interfaces (RESTful APIs) for its services [41], and discuss those APIs.

RESTful services are currently one of the most popular trends in developing cloud services. RESTful APIs follow the SOA model and, therefore, are often used by web service-based software architectures via XML or JSON for integration purposes. They are consumed via an Internet browser or by web servers and are a very suitable solution for the remote consumption of data processing services by heterogeneous users. RESTful applications use HTTP methods (PUT, GET, POST, DELETE) to realize the four CRUD (Create/Read/Update/Delete) operations, and can be very powerful, especially when used in combination with JSON messages. JSON (Java Script Object Notation) is a lightweight data interchange format that specifies parameters in the key-value pair fashion. It is very easy for humans to understand and for machines to parse and process. The service layer can be easily described through the set of exposed RESTful APIs, which can be invoked by nodes to obtain predictor model updates or single responses to a classification request. In the proposed solution, such APIs adopt JSON to specify the parameters contained in the HTTP requests and responses exchanged among the cloud services and the mobile nodes. The following main APIs were envisioned:

- Classify packet: obtain a classification for a given packet, identified by means of a set of features;
- Classify flow: obtain a classification for an entire packet flow, identified by means of a set of features;
- Get Updated Model: obtain the new predictor model in a well-known format (e.g., Predictor Model Markup Language - PMML);
- Get Updated Configuration: obtain the new predictor model in form of a partial bitstream to load onto the device;
- Send flow: send the features related to a given flow (this actually increases the volume of data available to the service to enhance its model).

Each of the APIs has its own URL and HTTP method (GET, POST, PUT, or DELETE). In particular, the classify packet and classify flow, along with the send flow APIs adopt the POST method, to push information about the packets to classify, while the get updated configuration API uses the GET HTTP method. The get updated model API has been introduced to extend the approach also to cases of pure software systems willing to take advantage of the cloud distributed computation capabilities to perform high-accuracy traffic analysis tasks.

To perform an API call, the client running on the mobile device has to perform a HTTP request. Method parameters are passed in the URL query string or in the message body, depending on the HTTP method. API responses are returned as JSON objects, which typically include metadata (i.e., the response status code and error message, if any) and the actual response, which contains the actual data formatted in the form key:value.

5.4.1 Analysis on the distributed layers

In order to evaluate the performance of the service layer, a prototype application were realized. In particular the application provides some services, such as the acquisition of traffic flows from devices and the model update, and executes the business logic. The server interface collects packets and/or flows that were marked as suspicious by devices because unclassifiable, and launches a further analysis based on the latest created traffic model. If the suspicious flow or packet is recognized either malicious or non-malicious, it is discarded by the server and the device is notified about the classification result and is possibly updated with the latest model available. Otherwise, the involved traffic is stored in a queue for further analysis, in order to establish its nature, i.e., in order to establish whether it represents a real threat for mobile devices or not. This task can be accomplished by an expert teamwork, or by semi-automatic procedures, that are not discussed here because out of scope. New information about traffic behavior extends the original dataset available on the server so that new knowledge can be extracted, by means of a machine learning algorithm, and new updated models can be exploited for traffic analysis. Knowledge extraction needs high computational resources, because the involved dataset contains a huge amount of traffic data. Similarly, model updating on mobile devices is an energy and time consuming task, and should not be too frequent. Hence, the service layer needs a model update scheduling that has to take into account: (i) the model aging, (ii) the global amount of unclassified traffic and (iii) the amount of malicious packets and flows that are added to the initial dataset, i.e. which are not yet exploited for the generation of new models. In the prototype, there were considered three different model update policies to take into account such features.

The prototype application were realized using the Java language, as here the focus is not strictly on the timing performance, but only on the application behavior under different workload conditions. Hence, in order to simulate devices' network traffic, varying the

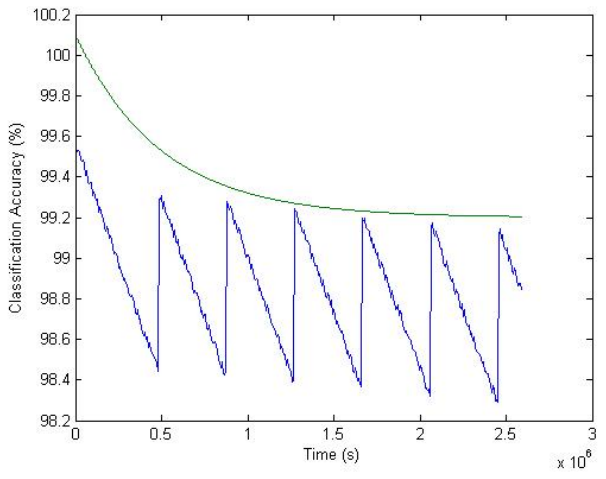


Figure 5.10: Analysis of the traffic classification accuracy reached thanks to dynamic reconfiguration and automatic generation of updated traffic models.

packets throughput, the devices population, the suspicious and malicious packets' and flows' probabilistic distribution, a multithreading approach were exploited. This setup exploited the same real network traffic dataset used in the previous experimental campaign (Subsection 5.3.4), partitioned into a learning set, used to generate the first traffic model, and a test set, that is used to feed the simulated mobile devices. In particular, to carry out the simulation experiments, 50 threats were picked over the available 150 threats to build the learning set, and used the remainder to simulate a real workload environment. As for the data traffic distribution, the threats arrival were modeled as a bathtub curve distribution, which is quite close to a typical distribution of application vulnerabilities: as indeed, after the first release the vulnerabilities reach the highest peak, then they decrease with subsequent software upgrades and patches, and finally they increase due to the application aging and to the increasing of attackers' skills. The distribution for the non malicious, but suspicious, traffic was uniform, since it includes traffic generated by new applications or new released Internet protocols.

As anticipated, the traffic model processed by the service layer is updated according to a scheduling policy that evaluates the first applicable condition among the following: (i) the model aging threshold exceeds 4 days, (ii) the number of new traffic flows/packets inserted in the analyzer queue exceeds 2000, (iii) the number of new detected threats exceeds 10. The simulations a time interval of about 1 month and let the device population and threat distribution vary to measure accuracy. In particular, Figure 5.10 shows how the traffic analysis accuracy varies over time for a large network of 6 million devices. As shown, due to the chosen probability distribution, the accuracy (blue curve) presents a generally decreasing trend. Moreover, accuracy peaks can be observed in correspondence to update events, thanks to the acquisition of new knowledge about traffic. Note that the time between two subsequent updates is not constant due to the different scheduling policies that are activated by deployed application. Clearly, the accuracy does not reach the maximum possible value at each update because of the delay experienced in the update process: when one of the three policies is triggered, a learning task is scheduled on the server, but since its execution takes a certain amount of time, in this interval new traffic information is not taken in to account. Hence, when the new traffic model is released, the evaluated accuracy does not achieve the original values. The peaks' trend (green curve) is exponentially decreasing to a stable value around 99.6%.

Chapter 6

Conclusion and Future Directions

Nowadays, an extremely wide spectrum of application domains adopts the FPGAs as in-field device technology. Numerous advantages, mainly related to the low cost and to the short time-to-market, have been characterizing the FPGA design methodology, such that they are competitors of the ASIC devices, since modern FPGAs are practically suitable to design just about anything. But, unlike the ASICs, FPGAs suffer from some weaknesses due to the inherently in-field (re)configuration feature. As indeed, they are much closer in nature to the software domain than to the mere hardware development, since each FPGA device can be programmed directly by end users providing a bitstream file, which is obtained by a design flow, similarly to the ASIC projects. Moreover, the FPGA design methodology is characterized by an extensive adoption of components-of-the-shelf, namely IP cores. Vulnerabilities of FPGAs jeopardize the IP core rights and the application which are deployed on them.

In this doctoral thesis, I have proposed some techniques to manage the rights of IP cores, protecting the bitstreams against known attacks. In particular, exploiting the partial reconfiguration and encryption features of modern FPGAs, I have identified parties involved in the IPs distribution process, the requirements for a secure distribution infrastructure and some improvements which can be included in future FPGA designs. Moreover, even targeting old and low-end FPGAs, I have demonstrated that PUFs can be adopted to guarantee a minimal security mechanism to bind a bitstream to a specific device by exploiting two architectures: the Anderson PUF and an innovative approach, called FS PUF. At the end, through an extensive case study, I have shown that the

FPGA is suitable to perform algorithms that require a very high throughput and I have envisioned a two-tier service-based infrastructure, which can automatically produce and securely distribute hardware updates.

With respect to the questions raised in the Section 1.1, I have replied each question giving the following contributions:

1. FPGA technology is characterized by several vulnerabilities, hence some attacks can succeed on the programmable devices. This doctoral thesis has taken into account such weaknesses with respect to the technological evolution. In particular, the readback attack exploits the readback feature of FPGAs to read the payload of the original bitstream used to configure them. By rebuilding the original bitstream, adding what is missing in the readback one, an attacker is able to clone the device. The cloning attack can be also accomplished by eavesdropping the bitstream file directly provided to the FPGA during the configuration process. Reverse engineering attack is able to extract from the bitstream the original netlist used to obtain it or some sensitive data, such as cryptography keys. SCA performed on FPGAs is able to extract secret information by exploiting the external observable phenomena of electronic devices, unintentionally produced by internal operations. All these attacks are able to threaten the application mission, divulging sensitive data that can compromise also third-parties IP cores. All the details have been reported in the Chapter 1

2. One of the most interesting FPGA feature is the reprogrammability since it enables the in-field device to receive hardware updates, pretty much like as the software case. In such a scenario, adopting a well-known DRM scheme, I have identify the roles which are crucial in the distribution process and, with the aim of protecting (partial) bitstreams, their concerns. The first identifiable role is the owner of the device, namely the End User. The End User requires the device to be secure and the other parties want to protect themselves against the End User attack attempts. The End User also is a part of the licensing process when it becomes the owner of the system and wants to bill new HDCs. The manufacturer of the FPGA chip within the device can be identified as FPGA Fabric Vendor and it guarantees some basic secure mechanisms on the devices in order to provide integrity and confidentiality. Its manufactured devices should not be equipped with backdoors.

The design and the implementation of the intellectual property as HDC is done by the IPCore Vendor, which distributes them as bitstreams. The IPCore Vendor is aware of the requirements that it has to fulfill, such as the compliance with the IP design specification and with the End User system. At the end, the TTP is an organization that all parties rely on, which guarantees that all the transactions between the parties are securely completed.

3. Since DRM mechanisms rely on cryptography primitives, which aim is to secure the involved IPs and associated rights, the FPGA technology has to provide built-in cryptography functions. In particular, a secure configuration mode should be available, where the configuration bitstream is stored outside the FPGA device in an encrypted form and it is only decrypted inside the FPGA within a secure perimeter. Then, readback of the plain bitstream has to be inhibited. Recently, the Xilinx has released some FPGA families which can be configured by means of enciphered bitstreams, even with partial reconfiguration, exploiting inherently AES decryption circuit and user-defined keys. The self-reconfiguration mechanism, together with all the DRM related functions, has been integrated within the Google Android Operating System by means of APIs. More details have been illustrated in Section 3.3.

4. PUFs architecture can be an alternative of the cryptographic mechanisms, which are not provided on old and low-end FPGA devices. In particular, it is possible to bind a bitstream such that the user design is able to work only on a specific FPGA device. Of course, without any protection on the content of the bitstream, it is still susceptible to some attacks, such as the reverse engineer. Since not all the devised PUFs architecture can be synthesized on such a kind of FPGA, I have investigated two different architectures. The first one is the Anderson PUF, which is scalable and a very reliable architecture. It was specifically designed for Xilinx devices, originally developed for the Xilinx Virtex-5. This thesis has illustrated an Anderson PUF implemented on the Spartan-3E technology, giving a deep analysis of some quality parameters and comparing them with other available Anderson PUF architectures. Then, I have explored the possibility to use the ring oscillators (ROs) to define a PUF based on measured frequencies, since the RO is an universal primitive, available on any silicon technology. Unlike the ROPUF, I have illustrated the frequencies signature PUF (FS PUF), which is ideally immune

to any disturbance effect provoked by uncontrolled working conditions. The FS PUF obeys to a statistical model and, by exploiting Xilinx Spartan-6 devices, it has been proved effective in discriminating a huge device population with few ROs.

5. FPGA is suitable for the execution of high throughput algorithms. To combine such feature with the reconfigurability, I have illustrated a case study in which the in-field application is a mobile end-user device equipped with an FPGA which runs a traffic analysis. Exploiting the decision tree machine learning algorithm, I have detailed a technique to automatically train and produce hardware traffic models, formally demonstrating the scalability of the approach. Such models are distributed to a devices population exploiting a two-tier service-based infrastructure, which is able to collect information about new traffic patterns and to automatically disseminate new generated hardware analyzers as partial bitstreams.

Even if the discussion in this thesis has been extensive and has covered a significant amount of aspects, there are still open issues that should be addressed. Indeed, as part of future work, the research activity can be expanded adopting the PUFs to define pay-per-use mechanism based on the inherently challenge-response mechanism. Indeed, being hard to predict, model and clone, the CRPs set is able to support a stronger DRM mechanism than the one illustrated in this doctoral thesis. As for the PUFs architectures, the Anderson PUFs need to be analyzed also with other uncertainty sources, such as the voltage and the aging, in order to better estimate the reliability which characterizes this PUF architecture. The FSPUF, introduced the first time in this thesis, should be tested by using more devices and, moreover, its efficacy has to be evaluated on the ASIC technology. At the end, as for the case study, which has involved the decision tree algorithm implemented on hardware, other predictors can be exploited to enhance not only the accuracy of the analysis, but also to minimize the area of the synthesized hardware accelerator. For instance, a multi-classification algorithm can be adopted to augment the classification accuracy and to try to reduce the amount of resource needed to synthesize the predictor in hardware.

Bibliography

[1] Ieee standard test access port and boundary scan architecture. *IEEE Std 1149.1-2001*, pages 1–212, July 2001. doi: 10.1109/IEEESTD.2001.92950.

[2] Widevine home page, 2014. URL `http://www.widevine.com/wv_drm.html`.

[3] Yousra Alkabani and Farinaz Koushanfar. Active hardware metering for intellectual property protection and security. In *USENIX Security*, pages 291–306, 2007.

[4] Open Mobile Alliance. Drm specification v2. 0. *Open Mobile Alliance Ltd*, 2004.

[5] Flora Amato, Mario Barbareschi, Valentina Casola, Antonino Mazzeo, and Sara Romano. Towards automatic generation of hardware classifiers. In *Algorithms and Architectures for Parallel Processing*, pages 125–132. Springer, 2013.

[6] Flora Amato, Mario Barbareschi, Valentina Casola, and Antonino Mazzeo. An fpga-based smart classifier for decision support systems. In *Intelligent Distributed Computing VII*, pages 289–299. Springer International Publishing, 2014.

[7] A. Amouri, F. Bruguier, S. Kiamehr, P. Benoit, L. Torres, and M. Tahoori. Aging effects in fpgas: an experimental analysis. In *Field Programmable Logic and Applications (FPL), 2014 24th International Conference on*, pages 1–4, Sept 2014. doi: 10.1109/FPL.2014.6927390.

[8] Jason H Anderson. A puf design for secure fpga-based embedded systems. In *Proceedings of the 2010 Asia and South Pacific Design Automation Conference*, pages 1–6. IEEE Press, 2010.

[9] James M Apland, David D Eaton, and Andrew K Chan. Security antifuse that prevents readout of some but not other information from a programmed field programmable gate array, April 27 1999. US Patent 5,898,776.

[10] Anshul Arora, Shree Garg, and Sateesh K. Peddoju. Malware detection using network traffic analysis in android based mobile devices. In *Next Generation Mobile Apps, Services and Technologies (NGMAST), 2014 Eighth International Conference on*, pages 66–71, Sept 2014. doi: 10.1109/NGMAST.2014.57.

[11] Khalil Arshak, E. Jafer, and C. Ibala. Testing fpga based digital system using xilinx chipscope logic analyzer. In *Electronics Technology, 2006. ISSE '06. 29th International Spring Seminar on*, pages 355–360, May 2006. doi: 10.1109/ISSE.2006.365129.

[12] M. Barbareschi, A. Mazzeo, and A. Vespoli. Un laboratorio elettronico su smartphone per dispositivi a microcontrollore. *Mondo Digitale*, 13(51):207–215, 2014.

[13] Mario Barbareschi, Ermanno Battista, Valentina Casola, and Nicola Mazzocca. On the adoption of fpga for protecting cyber physical infrastructures. In *P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2013 Eighth International Conference on*, pages 430–435. IEEE, 2013.

[14] Mario Barbareschi, Antonino Mazzeo, and Antonino Vespoli. Network traffic analysis using android on a hybrid computing architecture. In *Algorithms and Architectures for Parallel Processing*, pages 141–148. Springer International Publishing, 2013.

[15] Mario Barbareschi, Ermanno Battista, Antonino Mazzeo, and Sridhar Venkatesan. Advancing wsn physical security adopting tpm-based architectures. In *Information Reuse and Integration (IRI), 2014 IEEE 15th International Conference on*, pages 394–399. IEEE, 2014.

[16] Mario Barbareschi, Ermanno Battista, Nicola Mazzocca, and Sridhar Venkatesan. A hardware accelerator for data classification within the sensing infrastructure. In *Information Reuse and Integration (IRI), 2014 IEEE 15th International Conference on*, pages 400–405. IEEE, 2014.

[17] Mario Barbareschi, Alessandra De Benedictis, Antonino Mazzeo, and Antonino Vespoli. Mobile traffic analysis exploiting a cloud infrastructure and hardware accelerators. In *P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2014 Ninth International Conference on*, pages 414–419. IEEE, 2014.

[18] Mario Barbareschi, Pierpaolo Bagnasco, and Antonino Mazzeo. Quality trends analysis for the anderson puf varying the supplied voltage. In *Design & Technology of Integrated Systems In Nanoscale Era (DTIS), 2015 10th IEEE International Conference On*. IEEE, 2015.

[19] Mario Barbareschi, Ermanno Battista, Antonino Mazzeo, and Nicola Mazzocca. Testing 90nm microcontroller sram puf quality. In *Design & Technology of Integrated Systems In Nanoscale Era (DTIS), 2015 10th IEEE International Conference On*. IEEE, 2015.

[20] Mario Barbareschi, Alessandra De Benedictis, Antonino Mazzeo, and Antonino Vespoli. Providing mobile traffic analysis as-a-service: design of a service-based infrastructure to offer high-accuracy traffic classifiers based on hardware accelerators. *(In press) Journal of Digital Information Management (JDIM)*, 2015.

[21] Mario Barbareschi, Salvatore Del Prete, Francesco Gargiulo, Antonino Mazzeo, and Carlo Sansone. Decision tree-based multiple classifier systems: an fpga perspective. In *Multiple Classifier Systems*. Springer, 2015.

[22] Mario Barbareschi, Antonino Mazzeo, and Pierpaolo Bagnasco. Implementing reliable mechanisms for ip protection on low-end fpga devices, March 2015. URL `http://www.date-conference.com/conference/workshop-w10`. DATE W10 TRUDEVICE 2015, online publication.

[23] Mario Barbareschi, Antonino Mazzeo, and Antonino Vespoli. Malicious traffic analysis on mobile devices: a hardware solution. *(In press) International Journal of Big Data Intelligence*, 2(2), 2015.

[24] Mario Barbareschi, Lionel Torres, and Giorgio Di Natale. Ring oscillators analysis for fpga security purposes, March 2015. URL `http://www.date-conference.com/conference/workshop-w10`. DATE W10 TRUDEVICE 2015, online publication.

[25] Florian Benz, André Seffrin, and Sorin A Huss. Bil: A tool-chain for bitstream reverse-engineering. In *Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on*, pages 735–738. IEEE, 2012.

[26] Andrey Bogdanov, Amir Moradi, and Tolga Yalcin. Efficient and side-channel resistant authenticated encryption of fpga bitstreams. In *ReConFig*, pages 1–6, 2012.

[27] Vincent Carlier, Hervé Chabanne, Emmanuelle Dottax, and Hervé Pelletier. Electromagnetic side channels of an fpga implementation of aes. In *CRYPTOLOGY EPRINT ARCHIVE, REPORT 2004/145*. Citeseer, 2004.

[28] Hakima Chaouchi. *The internet of things: connecting objects*. John Wiley & Sons, 2013.

[29] Alessandro Cilardo, Mario Barbareschi, and Antonino Mazzeo. Secure distribution infrastructure for hardware digital contents. *IET Computers & Digital Techniques*, 8(6):300–310, 2014.

[30] Katherine Compton and Scott Hauck. Reconfigurable computing: a survey of systems and software. *ACM Computing Surveys (csuR)*, 34(2):171–210, 2002.

[31] Nathaniel Couture and Kenneth B Kent. Periodic licensing of fpga based intellectual property. In *Field Programmable Technology, 2006. FPT 2006. IEEE International Conference on*, pages 357–360. IEEE, 2006.

[32] Glenn Crow. Advanced security schemes for spartan-3a/3an/3a dsp fpgas. *Xilinx Corp. White Paper, ref*, 267, 2007.

[33] Jayita Das, Kevin Scott, Drew Burgett, Srinath Rajaram, and Sanjukta Bhanja. A novel geometry based mram puf. In *Nanotechnology (IEEE-NANO), 2014 IEEE 14th International Conference on*, pages 859–863. IEEE, 2014.

[34] Elke De Mulder, Pieter Buysschaert, SB Ors, Peter Delmotte, Bart Preneel, Guy Vandenbosch, and Ingrid Verbauwhede. Electromagnetic analysis attack on an fpga implementation of an elliptic curve cryptosystem. In *Computer as a Tool, 2005. EUROCON 2005. The International Conference on*, volume 2, pages 1879–1882. IEEE, 2005.

[35] Android Developers. What is android, 2011.

[36] Florian Devic, Lionel Torres, Jérémie Crenne, Benoit Badrignans, and Pascal Benoit. Secure dpr: Secure update preventing replay attacks for dynamic partial reconfiguration. In *Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on*, pages 57–62. IEEE, 2012.

[37] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM journal on computing*, 38(1):97–139, 2008.

[38] Saar Drimer, Tim Güneysu, Markus G Kuhn, and Christof Paar. Protecting multiple cores in a single fpga design. *Draft available at http://www. cl. cam. ac. uk/sd410/, written May*, 2008.

[39] Ahmet M Eskicioglu, John Town, and Edward J Delp III. Security of digital entertainment content from creation to consumption. In *International Symposium on Optical Science and Technology*, pages 187–211. International Society for Optics and Photonics, 2001.

[40] Hossein Falaki, Dimitrios Lymberopoulos, Ratul Mahajan, Srikanth Kandula, and Deborah Estrin. A first look at traffic on smartphones. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 281–287. ACM, 2010.

[41] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000.

[42] V Fischer and F Bernard. True random number generators in fpgas. In *Security Trends for FPGAS*, pages 101–135. Springer, 2011.

[43] Arik Friedman and Assaf Schuster. Data mining with differential privacy. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge discovery and data mining*, pages 493–502, New York, NY, USA, 2010. ACM.

[44] Blaise Gassend, Dwaine Clarke, Marten Van Dijk, and Srinivas Devadas. Silicon physical random functions. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 148–160. ACM, 2002.

[45] Sezer Gören, Ozgur Ozkurt, Abdullah Yildiz, H Fatih Ugurdag, Rajat S Chakraborty, and Debdeep Mukhopadhyay. Partial bitstream protection for low-cost fpgas with physical unclonable function, obfuscation, and dynamic partial self reconfiguration. *Computers & Electrical Engineering*, 39(2):386–397, 2013.

[46] Paul Graham, Brent Nelson, and Brad Hutchings. Instrumenting bitstreams for debugging fpga circuits. In *Field-Programmable Custom Computing Machines, 2001. FCCM'01. The 9th Annual IEEE Symposium on*, pages 41–50. IEEE, 2001.

[47] TC Group et al. TCG specification architecture overview revision 1.2, 2004.

[48] Jorge Guajardo, Sandeep S Kumar, G-J Schrijen, and Pim Tuyls. Physical unclonable functions and public-key crypto for fpga ip protection. In *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*, pages 189–195. IEEE, 2007.

[49] Jorge Guajardo, Sandeep S Kumar, Geert-Jan Schrijen, and Pim Tuyls. *FPGA intrinsic PUFs and their use for IP protection.* Springer, 2007.

[50] T Guneysu, Bodo Moller, and Christof Paar. Dynamic intellectual property protection for reconfigurable devices. In *Field-Programmable Technology, 2007. ICFPT 2007. International Conference on*, pages 169–176. IEEE, 2007.

[51] Scott Hauck and Andre DeHon. *Reconfigurable computing: the theory and practice of FPGA-based computation.* Morgan Kaufmann, 2010.

[52] Yohei Hori, Akashi Satoh, Hirofumi Sakane, and Kenji Toda. Bitstream encryption and authentication with aes-gcm in dynamically reconfigurable systems. In *Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on*, pages 23–28. IEEE, 2008.

[53] Yohei Hori, Takahiro Yoshida, Toshihiro Katashita, and Akashi Satoh. Quantitative and statistical performance evaluation of arbiter physical unclonable functions on fpgas. In *Reconfigurable Computing and FPGAs (ReConFig), 2010 International Conference on*, pages 298–303. IEEE, 2010.

[54] Miaoqing Huang and Shiming Li. A delay-based puf design using multiplexers on fpga. In *Field-Programmable Custom Computing Machines (FCCM), 2013 IEEE 21st Annual International Symposium on*, pages 226–226. IEEE, 2013.

[55] G Joy Persial, M Prabhu, and R Shanmugalakshmi. Side channel attack-survey. *Int J Adva Sci Res Rev*, 1(4):54–57, 2011.

[56] Heiko Kalte, Dominik Langen, Erik Vonnahme, André Brinkmann, and U Ruckert. Dynamically reconfigurable system-on-programmable-chip. In *Parallel, Distributed and Network-based Processing, 2002. Proceedings. 10th Euromicro Workshop on*, pages 235–242. IEEE, 2002.

[57] Deniz Karakoyunlu and Berk Sunar. Differential template attacks on puf enabled cryptographic devices. In *Information Forensics and Security (WIFS), 2010 IEEE International Workshop on*, pages 1–6. IEEE, 2010.

[58] Tom Kean. Secure configuration of field programmable gate arrays. In *Field-Programmable Logic and Applications*, pages 142–151. Springer, 2001.

[59] Krzysztof Kepa, Fearghal Morgan, Krzysztof Kosciuszkiewicz, and Tomasz Surmacz. Serecon: A secure dynamic partial reconfiguration controller. In *Symposium on VLSI, 2008. ISVLSI'08. IEEE Computer Society Annual*, pages 292–297. IEEE, 2008.

[60] HyunHo Kim, Ndibanje Bruce, Hoon-Jae Lee, YongJe Choi, and Dooho Choi. Side channel attacks on cryptographic module: Em and pa attacks accuracy analysis. In *Information Science and Applications*, pages 509–516. Springer, 2015.

[61] Paul C Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Advances in Cryptology—CRYPTO'96*, pages 104–113. Springer, 1996.

[62] Sandeep Kumar, Christof Paar, Jan Pelzl, Gerd Pfeiffer, and Manfred Schimmler. Breaking ciphers with copacobana–a cost-optimized parallel code breaker. In *Cryptographic Hardware and Embedded Systems-CHES 2006*, pages 101–118. Springer, 2006.

[63] Sandeep S Kumar, Jorge Guajardo, Roel Maes, G-J Schrijen, and Pim Tuyls. The butterfly puf protecting ip on every fpga. In *Hardware-Oriented Security and Trust, 2008. HOST 2008. IEEE International Workshop on*, pages 67–70. IEEE, 2008.

[64] Jae W Lee, Daihyun Lim, Blaise Gassend, G Edward Suh, Marten Van Dijk, and Srinivas Devadas. A technique to build a secret key in integrated circuits for identification and authentication applications. In *VLSI Circuits, 2004. Digest of Technical Papers. 2004 Symposium on*, pages 176–179. IEEE, 2004.

[65] A Lesea. jbits & reverse engineering (usenet comp. arch. fpga), september 2005.

[66] Daihyun Lim, Jae W Lee, Blaise Gassend, G Edward Suh, Marten Van Dijk, and Srinivas Devadas. Extracting secret keys from integrated circuits. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 13(10):1200–1205, 2005.

[67] Yeon-sup Lim, Hyun-chul Kim, Jiwoong Jeong, Chong-kwon Kim, Ted Taekyoung Kwon, and Yanghee Choi. Internet traffic classification demystified: on the sources of the discriminative power. In *Proceedings of the 6th International COnference*, page 9. ACM, 2010.

[68] Dominik Lorenz, Georg Georgakos, and Ulf Schlichtmann. Aging analysis of circuit timing considering nbti and hci. In *On-Line Testing Symposium, 2009. IOLTS 2009. 15th IEEE International*, pages 3–8. IEEE, 2009.

[69] Hongwei Luo, Guili He, Xiaodong Lin, and Xuemin Shen. Towards hierarchical security framework for smartphones. In *Communications in China (ICCC), 2012 1st IEEE International Conference on*, pages 214–219, 2012. doi: 10.1109/ICCChina.2012.6356880.

[70] Roel Maes and Ingrid Verbauwhede. Physically unclonable functions: A study on the state of the art and future research directions. In *Towards Hardware-Intrinsic Security*, pages 3–37. Springer, 2010.

[71] Roel Maes, Pim Tuyls, and Ingrid Verbauwhede. Intrinsic pufs from flip-flops on reconfigurable devices. In *3rd Benelux workshop on information and system security (WISSec 2008)*, volume 17, 2008.

[72] Roel Maes, Dries Schellekens, and Ingrid Verbauwhede. A pay-per-use licensing scheme for hardware ip cores in recent sram-based fpgas. *Information Forensics and Security, IEEE Transactions on*, 7(1):98–108, 2012.

[73] Ahmed Mahmoud, Ulrich Rührmair, Mehrdad Majzoobi, and Farinaz Koushanfar. Combined modeling and side channel attacks on strong pufs. *IACR Cryptology ePrint Archive*, 2013:632, 2013.

[74] Abhranil Maiti and Patrick Schaumont. Improved ring oscillator puf: An fpga-friendly secure primitive. *Journal of cryptology*, 24(2):375–397, 2011.

[75] Abhranil Maiti, Jeff Casarona, Luke McHale, and Patrick Schaumont. A large scale characterization of ro-puf. In *Hardware-Oriented Security and Trust (HOST), 2010 IEEE International Symposium on*, pages 94–99. IEEE, 2010.

[76] Andrei Marghescu, George Teseleanu, Diana-Stefania Maimut, Traian Neacsa, and Paul Svasta. Adapting a ring oscillator-based true random number generator for zynq system on chip embedded platform. In *Design and Technology in Electronic Packaging (SIITME), 2014 IEEE 20th International Symposium for*, pages 197–202. IEEE, 2014.

[77] Dominik Merli, Frederic Stumpf, and Claudia Eckert. Improving the quality of ring oscillator pufs on fpgas. In *Proceedings of the 5th Workshop on Embedded Systems Security*, page 9. ACM, 2010.

[78] Dominik Merli, Dieter Schuster, Frederic Stumpf, and Georg Sigl. Semi-invasive em attack on fpga ro pufs and countermeasures. In *Proceedings of the Workshop on Embedded Systems Security*, page 2. ACM, 2011.

[79] Dominik Merli, Dieter Schuster, Frederic Stumpf, and Georg Sigl. Side-channel analysis of pufs and fuzzy extractors. In *Trust and Trustworthy Computing*, pages 33–47. Springer, 2011.

[80] Dominik Merli, Johann Heyszl, Benedikt Heinz, Dieter Schuster, Frederic Stumpf, and Georg Sigl. Localized electromagnetic analysis of ro pufs. In *Hardware-Oriented Security and Trust (HOST), 2013 IEEE International Symposium on*, pages 19–24. IEEE, 2013.

[81] Noman Mohammed, Rui Chen, Benjamin C.M. Fung, and Philip S. Yu. Differentially private data release for data mining. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 493–501, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0813-7.

[82] Alireza Monemi, Roozbeh Zarei, and Muhammad N Marsono. Online NetFPGA Decision Tree Statistical Traffic Classifier. *Computer Communications*, 2013.

[83] Amir Moradi, Alessandro Barenghi, Timo Kasper, and Christof Paar. On the vulnerability of fpga bitstream encryption against power analysis attacks: extracting keys from xilinx virtex-ii fpgas. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 111–124. ACM, 2011.

[84] Amir Moradi, Markus Kasper, and Christof Paar. On the portability of side-channel attacks–an analysis of the xilinx virtex 4, virtex 5, and spartan 6 bitstream encryption mechanism–. 2011.

[85] Amir Moradi, David Oswald, Christof Paar, and Pawel Swierczynski. Side-channel attacks on the bitstream encryption mechanism of altera stratix ii: facilitating black-box analysis using software reverse-engineering. In *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*, pages 91–100. ACM, 2013.

[86] S Mühlbach and Andreas Koch. A novel network platform for secure and efficient malware collection based on reconfigurable hardware logic. In *Internet Security (WorldCIS), 2011 World Congress on*, pages 9–14. IEEE, 2011.

[87] David Naccache and Patrice Fremanteau. Unforgeable identification device, identification device reader and method of identification, July 18 1995. US Patent 5,434,917.

[88] Cisco Visual Networking. Cisco visual networking index: Global mobile data traffic forecast update, 2014–2019. *Cisco white paper*, 2015.

[89] Jean-Baptiste Note and Éric Rannaud. From the bitstream to the netlist. In *FPGA*, volume 8, pages 264–264, 2008.

[90] Sıddıka Berna Örs, Elisabeth Oswald, and Bart Preneel. Power-analysis attacks on an fpga–first experimental results. In *Cryptographic Hardware and Embedded Systems-CHES 2003*, pages 35–50. Springer, 2003.

[91] Eric Peeters, François-Xavier Standaert, Nicolas Donckers, and Jean-Jacques Quisquater. Improved higher-order side-channel attacks with fpga experiments.

In *Cryptographic Hardware and Embedded Systems–CHES 2005*, pages 309–323. Springer, 2005.

[92] Gang Qu and Chi-En Yin. Temperature-aware cooperative ring oscillator puf. In *Hardware-Oriented Security and Trust, 2009. HOST'09. IEEE International Workshop on*, pages 36–42. IEEE, 2009.

[93] J. Ross Quinlan. *C4.5: programs for machine learning.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993. ISBN 1-55860-238-0.

[94] Pappu Srinivasa Ravikanth. *Physical one-way functions.* PhD thesis, Massachusetts Institute of Technology, 2001.

[95] William Rosenblatt, Stephen Mooney, and William Trippe. *Digital rights management: business and technology.* M&T Books, 2003.

[96] Ulrich Rührmair, Frank Sehnke, Jan Sölter, Gideon Dror, Srinivas Devadas, and Jürgen Schmidhuber. Modeling attacks on physical unclonable functions. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 237–249. ACM, 2010.

[97] Saber Salah, Sami Abduljalil Abdulhak, Hyontai Sug, Dae-Ki Kang, and HoonJae Lee. Performance analysis of intrusion detection systems for smartphone security enhancements. In *Mobile IT Convergence (ICMIC), 2011 International Conference on*, pages 15–19. IEEE, 2011.

[98] Lester Sanders. Secure boot of zynq-7000 all programmable soc. *Application note XAPP1175 (v1. 0), Xilinx*, 2013.

[99] Eric E Schadt, Michael D Linderman, Jon Sorenson, Lawrence Lee, and Garry P Nolan. Computational solutions to large-scale data management and analysis. *Nature Reviews Genetics*, 11(9):647–657, 2010.

[100] Pete Sedcole and Peter YK Cheung. Within-die delay variability in 90nm fpgas and beyond. In *Field Programmable Technology, 2006. FPT 2006. IEEE International Conference on*, pages 97–104. IEEE, 2006.

[101] P Skoda, B Medved Rogina, and V Sruk. FPGA implementations of data mining algorithms. In *MIPRO, 2012 Proceedings of the 35th International Convention*, pages 362–367. IEEE, 2012.

[102] Sergei Skorobogatov and Christopher Woods. *Breakthrough silicon scanning discovers backdoor in military chip*. Springer, 2012.

[103] Sergei Petrovich Skorobogatov. *Semi-invasive attacks: a new approach to hardware security analysis*. PhD thesis, University of Cambridge Ph. D. dissertation, 2005.

[104] Maureen Smerdon. Security solutions using spartan-3 generation fpgas. In *Xilinx Inc*. Citeseer, 2008.

[105] F-X Standaert, François Macé, Eric Peeters, and J-J Quisquater. Updates on the security of fpgas against power analysis attacks. In *Reconfigurable Computing: Architectures and Applications*, pages 335–346. Springer, 2006.

[106] François-Xavier Standaert, Loïc van Oldeneel tot Oldenzeel, David Samyde, and Jean-Jacques Quisquater. Power analysis of fpgas: How practical is the attack? In *Field Programmable Logic and Application*, pages 701–710. Springer, 2003.

[107] François-Xavier Standaert, Sıddıka Berna Örs, Jean-Jacques Quisquater, and Bart Preneel. Power analysis attacks against fpga implementations of the des. In *Field Programmable Logic and Application*, pages 84–94. Springer, 2004.

[108] O-X Standaert, Éric Peeters, Gaël Rouvroy, and J-J Quisquater. An overview of power analysis attacks against field programmable gate arrays. *Proceedings of the IEEE*, 94(2):383–394, 2006.

[109] G Edward Suh and Srinivas Devadas. Physical unclonable functions for device authentication and secret key generation. In *Proceedings of the 44th annual Design Automation Conference*, pages 9–14. ACM, 2007.

[110] Mohammad Tehranipoor and Farinaz Koushanfar. A survey of hardware trojan taxonomy and detection. 2010.

[111] Anil Telikepalli and I Xilinx. Is your fpga design secure. *XCell Journal, XILINX, Fall*, 2003.

[112] Tran Thanh, Vu Huu Tiep, Tran Hoang Vu, Pham Ngoc Nam, and Nguyen Van Cuong. Secure remote updating of bitstream in partial reconfigurable embedded systems based on fpga. In *Computing, Management and Telecommunications (ComManTel), 2013 International Conference on*, pages 152–156, Jan 2013. doi: 10.1109/ComManTel.2013.6482382.

[113] Kris Tiri and Ingrid Verbauwhede. Synthesis of secure fpga implementations. *IACR Cryptology ePrint Archive*, 2004:68, 2004.

[114] Da Tong, Lu Sun, Kiran Matam, and Viktor Prasanna. High throughput and programmable online traffic classifier on FPGA. In *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*, pages 255–264. ACM, 2013.

[115] Altera Verification Tool. Signaltap ii embedded logic analyzer, 2006.

[116] Steve Trimberger. Trusted design in fpgas. In *Proceedings of the 44th annual Design Automation Conference*, pages 5–8. ACM, 2007.

[117] Smith Tsang, Ben Kao, Kevin Y Yip, Wai-Shing Ho, and Sau Dan Lee. Decision trees for uncertain data. *Knowledge and Data Engineering, IEEE Transactions on*, 23(1):64–78, 2011.

[118] Elena Ioana Vatajelu, Giorgio Di Natale, Marco Indaco, and Paolo Ernesto Prinetto. Stt mram-based pufs. In *Proceedings of Design, Automation and Test in Europe 2015*. IEEE, 2015.

[119] Florian Wilde, Matthias Hiller, and Michael Pehl. Statistic-based security analysis of ring oscillator pufs. In *Integrated Circuits (ISIC), 2014 14th International Symposium on*, pages 148–151. IEEE, 2014.

[120] *Spartan-6 FPGA Configurable Logic Block*. Xilinx. Available at `http://www.xilinx.com/support/documentation/user_guides/ug384.pdf`.

[121] Xilinx. Zynq-7000 ap soc redirecting ethernet packet to pl for hardware packet inspection tech tip, September 2013. URL `http://www.wiki.xilinx.com/Zynq-7000+AP+SoC+Redirecting+Ethernet+Packet+to+PL+for+Hardware+Packet+Inspection+Tech+Tip`.

[122] Jiliang Zhang, Yaping Lin, Yongqiang Lyu, Ray CC Cheung, Wenjie Che, Qiang Zhou, and Jinian Bian. Binding hardware ips to specific fpga device via intertwining the puf response with the fsm of sequential circuits. In *Field-Programmable Custom Computing Machines (FCCM), 2013 IEEE 21st Annual International Symposium on*, pages 227–227. IEEE, 2013.

[123] Jiliang Zhang, Yaping Lin, Yongqiang Lyu, Gang Qu, Ray CC Cheung, Wenjie Che, Qiang Zhou, and Jinian Bian. Fpga ip protection by binding finite state machine to physical unclonable function. In *Field Programmable Logic and Applications (FPL), 2013 23rd International Conference on*, pages 1–4. IEEE, 2013.

[124] Jiliang Zhang, Qiang Wu, Yongqiang Lyu, Qiang Zhou, Yici Cai, Yaping Lin, and Gang Qu. Design and implementation of a delay-based puf for fpga ip protection. In *Computer-Aided Design and Computer Graphics (CAD/Graphics), 2013 International Conference on*, pages 107–114. IEEE, 2013.

[125] Jiliang Zhang, Yaping Lin, Yongqiang Lyu, and Gang Qu. A puf-fsm binding scheme for fpga ip protection and pay-per-device licensing. 2015.

[126] Daniel Ziener, Stefan Aßmus, and Jürgen Teich. Identifying fpga ip-cores based on lookup table content analysis. In *Field Programmable Logic and Applications, 2006. FPL'06. International Conference on*, pages 1–6. IEEE, 2006.