POLITECNICO DI TORINO
Repository ISTITUZIONALE

Securing Critical Infrastructures

Original
Securing Critical Infrastructures / Carelli, Alberto. - (2020 Sep 03), pp. 1-167.

Availability:
This version is available at: 11583/2850592 since: 2020-10-30T11:25:26Z

Publisher:
Politecnico di Torino

Published
DOI:

Terms of use:
Altro tipo di accesso

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

27 June 2026

Doctoral Dissertation
Doctoral Program in Computer and Control Engineering (32nd cycle)

Securing Critical Infrastructures

Alberto Carelli

* * * * * *

Supervisor
Prof. Stefano Di Carlo, Supervisor

Doctoral Examination Committee:
Prof., Ramon Canal, UPC - Universitat Politècnica de Catalunya (Spain)
Prof., Luca Maria Cassano, Politecnico di Milano (Italy)
Prof., Giorgio Di Natale, CNRS - Centre national de la recherche scientifique (France)
Prof., Alberto Bosio, Ecole Centrale de Lyon (France)
Prof., Marco Torchiano, Politecnico di Torino (Italy)

Politecnico di Torino
2020

This thesis is licensed under a Creative Commons License, Attribution - Noncommercial-NoDerivative Works 4.0 International: see www.creativecommons.org. The text may be reproduced for non-commercial purposes, provided that credit is given to the original author.

I hereby declare that, the contents and organisation of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

. . . . . .

Alberto Carelli

Turin, 2020

Summary

The growing threat of advanced cyber-attacks is a major cause of concerns for Information-Technology (IT) systems. Cybersecurity implies guaranteeing the security of the cyberspace from threats, which might present in different forms. Among the different classes of system that employ IT systems, there are the Critical Infrastructures. Critical infrastructures describe all physical and cyber systems, assets and other elements on which the society of a nation relies upon to maintain national security, economic vitality, and public health and safety. Examples of critical infrastructures for a nation can be power plants and energy supply networks, water supply systems, healthcare and hospital structures, transportation infrastructures, etc.

Originally Critical Infrastructures were designed as isolated systems not connected to the internet. They were based on old legacy systems, lacking the security protocols that are now built in. In recent years there has been a proliferation of new digital technologies that are designed to provides beneficial features to the systems where they are embedded. However, also the integration of such technologies makes the systems more vulnerable from a cybersecurity point of view by expanding the attack surface. The lack or inadequacy of appropriate security mechanisms leads to malicious attacks. If successful, attacks to these classes of systems, may lead to physical disruption or business operations and intellectual property theft. This may result in extensive economic losses and expose health or social well-being of people to safety and security risks.

This thesis focuses on the study of protection mechanisms for Critical Infrastructures from new cyberattacks. Different classes of attacks are considered.

Microarchitectural side-channel attacks exploit the microarchitecture of a microprocessor for unintended leakage of sensitive information. Targeting the computational cores, which are at the base of the majority of cyber assets, makes Critical Infrastructures and industrial control systems potentially vulnerable. Particular attention must be paid to the implementation of countermeasures against this type of attacks, because they might interfere with other crucial aspects of the system, such as safety.

Critical Infrastructures employ new technologies, such as reconfigurable platforms to exploit additional flexibility and acceleration capabilities. Also FPGA-based systems are vulnerable to attacks targeting the design to be deployed in-the-field. These attacks might target the confidentiality of the design or the injection of unintended features allowing the system to be controlled by malicious attackers or discover and exfiltrate sensitive information, such as intellectual property data. Although existing security measures already mitigate the attacks, they do not deal with a complex scenario where multiple different designers are involved.

Finally, a broader protection mechanism is provided through a hardware security module featuring secure key management and cryptographic functionalities. The robustness of the device is assessed through a physical and non-invasive side-channel attack aiming at the recovery of the stored encryption keys.

Acknowledgements

I express my sincere appreciation to those who have contributed during this journey. I cannot help but recognize the invaluable support of colleagues, family and friends. Without any of them, this research work would not have been possible.

First, I thank my Ph.D. advisor, Professor Stefano Di Carlo, for giving me the chance to continue on this long journey. I express my endless gratitude for his endless and continuous support. I thank him for his wisdom, for the patience he has shown and for teaching me the work of rigorous research. I value and admire every word spent and every precious advice received. But most important, I appreciate all the time he dedicated to me.

I acknowledge every member of my thesis committee for their effort and the time they dedicated to read my dissertation. In particular, I would like to thank the reviewers, Prof. Luca Maria Cassano and Prof. Ramon Canal, with their kind and insightful comments in the last few steps of this journey.

I am grateful to my family for the constant support provided. I thank them for their continuous presence, luckily starting long before the Ph.D. (and also before primary school). I thank them for their help and for the invaluable lessons non-academia-related that shaped me during all these years.

Heartfelt thanks goes to my fellow travellers who made my stay in Lab 6 more enjoyable. In no-particular-order: thanks to Roberta, for her understanding, insight and empathy shown along the travel. Thanks to Alessandro, for having my back and for the great help given while working together "against" the publications.

Thanks to Pascal, Andrea, Stefano, Francesco, Fabio, Teo, Juan, Alberto, Luisa, Lorenzo, Luigi, Irene, Marilena and Flavia for sharing your time with me during the lab-life. Thank you all for the happiness and the funny moments shared together.

I thank also the lab-neighbors, Evelina Di Corso, Ph.D. and her friends, who have always been kind to me. Particular thanks to Valerio Tenace, Ph.D. and Amirhosein Toosi, Ph.D. for their support and their encouragement (not only) in a difficult situation. Many thanks to Valentino Razza, Ph.D. for his help, for every

coffee offered and for the many "tips and tricks" of the job. I gratefully acknowledge my friend Paolo Viotti, Ph.D. for his advices and suggestions along the way.

Also my gratitude goes to thesis students and friends met here: Gheorghe, Lubin, Lorenzo, Nicola, Giulio, Edrian, Mauro e Carlo Alberto.

A special thank goes to my other comrades outside the school. Though dispersed throughout the world, we are still somehow connected...

a.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This chapter contains the description of the context of the research in the field of the security for Critical Infrastructures.

1.1 Critical Infrastructures

Critical infrastructures describe all physical and cyber systems that are crucial to a nation, i.e., their malfunction, incapacity or destruction would have an important debilitating impact on the physical or economic security, public health or safety of citizens of the nation involved. Critical infrastructures include also the assets, the systems, the facilities, the networks, and other elements on which the society relies upon to maintain national security, economic vitality, and public health and safety. Informally, critical infrastructures can be seen as the power used in homes, the water supply systems, the transportation service that connect cities, the various communication systems people rely on to maintain contact with others. Depending on each country, the Critical Infrastructures or part of them might be owned by the private sector or by the State itself. In the U.S.A., the elements of an infrastructure, either physical or cyber, are typically owned and operated by the private sector, though some belongs to federal, state, or local governments. Not every infrastructure can be considered critical to a nation or region. Thus, to formalize the definition, it is necessary to identify which infrastructure is both critical to maintain continued services or functions and vulnerable to some type of threat or hazard.

Although it is easy to perceive the sense and the importance of a Critical Infrastructure, there is no worldwide definition globally valid. Instead, several definitions do exist.

In the U.S.A., Critical Infrastructures are defined as "systems and assets, whether physical or virtual, so vital to the United States that the incapacity or destruction of such systems and assets would have a debilitating impact on security, national

economic security, national public health or safety, or any combination of those matters" [110].

In Europe, instead, there are two definitions:

1. *Critical Infrastructures* are defined as "assets, systems or parts thereof located in Member States, which are essential for the maintenance of vital societal functions, health, safety, security, economic or social well-being of people, and the disruption or destruction of which would have a significant impact in a Member State as a result of the failure to maintain those functions",

2. *European critical infrastructures (ECI)* are defined as "critical infrastructures located in Member States the disruption or destruction of which would have a significant impact on at least two Member States. The significance of the impactshall be assessed in terms of cross-cutting criteria. This includes effects resulting from cross-sector dependencies on other types of infrastructures".

The first definition refers to a Critical infrastructure within a Member State while the second definition refers to a Critical infrastructure considering the European Union as a whole. The above definitions are extracted from [70].

It is possible to see that the actual definition of Critical Infrastructure depends at least on the Region where it resides. Apart from U.S.A. and Europe, each sub-State or region might consider a different definition (e.g., each Member State within European Union has its own definition - which resemble the one of U.S.A.).

There are, however, generic definitions provided by international entities, such as:

1. ITU-T: "The key systems, services and functions whose disruption or destruction would have a debilitating impact on public health and safety, commerce, and national security, or any combination of these" [212];

2. NATO: "Physical or virtual systems and assets under the jurisdiction of a State that are so vital that their incapacitation or destruction may debilitate a State's security, economy, public health or safety, or the environment" [209];

3. IETF: "Those systems that are so vital to a nation that their incapacity or destruction would have a debilitating effect on national security, the economy, or public health and safety" [213].

Another discriminating characteristic is the operational field of the Critical Intrastructures. A sector can be categorized depending on the ownership:

- public ownership (also called State ownership or government ownership): the majority of the companies, the assets and the networks of a Critical Infrastructure is owned by the State;

- private ownership: the majority of the companies, the assets and the networks of a Critical Infrastructure is owned by a private group of individuals.

Not every sector is equally critical for the citizens. The Department of Homeland Security (DHS) of the United States of America considers 16 critical infrastructure sectors whose assets, systems, and networks, whether physical or virtual, are considered so vital to the United States that their incapacitation or destruction would have a debilitating effect on security, national economic security, national public health or safety, or any combination thereof. The sectors in which critical infrastructures are employed, reported for U.S.A. in Homeland Security Presidential Directive 7 [107], superseded by Presidential Policy Directive 21 (PPD-21) [111], can be summarized as follows.

Chemical Sector: The Chemical Sector encomprises several chemical facilities in a global supply chain and is able to convert various raw materials into several diverse products that are essential to modern life and to other supply chains. This sector can be in turn subdivided into five main segments: Basic chemicals, Specialty chemicals, Agricultural chemicals, Pharmaceuticals and Consumer products. Securing the treated chemicals against growing and evolving threats requires vigilance from both the public and private sector, given that the majority of Chemical Sector facilities are privately owned.

Commercial Facilities Sector: The Commercial Facilities Sector includes a diverse range of sites visited by several people or crowds for shopping, business, entertainment. In general the public can move freely within the site area. The majority of these facilities are privately owned and operated with minimal interaction required by the government.

Communications Sector: The Communications Sector underlies the operations of all businesses, organizations, and government. It is critical because it provides an "enabling function" across all critical infrastructure sectors. It provides communication services, which have evolved from voice services into an interconnected industry using terrestrial, satellite, and wireless transmission systems. The private sector, as owners and operators of the majority of communications infrastructure, is the primary entity responsible for protecting sector infrastructure and assets.

Critical Manufacturing Sector: The Critical Manufacturing Sector identifies several industries such as Metals Manufacturing, Machinery Manufacturing, Electrical Equipment, Appliance, and Component Manufacturing and Transportation Equipment Manufacturing. It is classified as critical because the products made by these manufacturing industries are essential to many other critical infrastructure sectors.

Dams Sector: The Dams Sector offers critical water retention and control services as well as hydroelectric power generation, water supply systems, agricultural irrigation, flood control, river navigation and industrial waste management. Also in this case, the services support multiple critical infrastructure sectors and industries. In the United States, the majority of the facilities (about 80%) are regulated by state dams safety offices.

Defense Industrial Base Sector: The Defense Industrial Base Sector is the industrial complex that enables research and development of the military systems, during the phases of design, production, delivery, and maintenance of military weapons systems, subsystems and components required by the military. The sector provides also products and services that are essential to mobilize, deploy, and sustain military operations.

Emergency Services Sector: The Emergency Services Sector (ESS) incorporates physical and cyber resources together with trained and skilled work force and is able to provide a wide range of prevention, preparedness, response, and recovery services during both day-to-day operations and incident response. The facilities are geographically distributed across the territory, at varios levels of government (e.g., federal, state, local, etc.). The ESS also includes private sector resources, such as industrial fire departments, private security organizations, and private emergency medical services providers. The organizations can be composed also by volunteers. Five distinct disciplines compose the ESS: Law Enforcement, Fire and Rescue Services, Emergency Medical Services, Emergency Management, Public Works. Finally, the ESS also provides specialized emergency services through individual personnel and teams (e.g., tactical teams, Search and Rescue Teams, National Guard Civil Support, etc.).

Energy Sector: The Energy Sector is considered as uniquely critical because it provides an "enabling function" across all critical infrastructure sectors. The energy infrastructure is divided into three interrelated segments: electricity, oil, and natural gas. In the U.S.A., the electricity segment contains more than 6,413 power plants. The reliance of virtually all industries on electric power and fuels means that all sectors have some dependence on the Energy Sector.

Financial Services Sector: The Financial Services Sector includes thousands of bank institutions, providers of investment products, insurance companies, other credit and financing organizations. Financial institutions vary widely in size and presence, ranging from some of the world's largest global companies with thousands of employees and many billions of dollars in assets, to

community banks and credit unions with a small number of employees serving individual communities. Thi sector represents a vital component of the nation's critical infrastructures.

Food and Agriculture Sector: The Food and Agriculture Sector is composed of farms, restaurants and other facilities that store, manufacture and process food. It has several critical dependencies in other sectors, but particularly with the Water and Wastewater Systems for clean irrigation and processed water, Transportation Systems for movement of products and livestock, the Energy Sector to power the equipment needed for agriculture production and food processing and the Chemical Sector for fertilizers and pesticides used in the production of crops. This sector is almost entirely under private ownership.

Government Facilities Sector: The Government Facilities Sector includes a wide variety of buildings, physically located on the territory of a nation and overseas. The buildings and the infrastructures are owned by the different levels of government (e.g., federal, state, local, etc.). Some of the facilities might be open to public (such as public offices), while others contain highly sensitive information, materials, processes or equipment (e.g., special-use military installations, embassies, courthouses, etc.). Apart from the physical infrastructures, the sector includes all the cyber elements that contribute to the protection of the sector assets.

Healthcare and Public Health Sector: The Healthcare and Public Health Sector protects all sectors of the economy from hazards such as terrorism, infectious disease outbreaks, and natural disasters. The sector plays a significant role in response and recovery across all other sectors in the event of a natural or manmade disaster. The assets of this sector are both private and public owned depending on the country. In both cases, the sector is managed across all levels of government.

Information Technology Sector: The Information Technology Sector is central to a country security, economy and safety, which are increansigly dependent on this sector's functions. These virtual and distributed functions produce and provide hardware, software, and information technology systems and services. Since the Internet represents the backbone of these infrastructures, this sector is tightly related with the Communication Sector. This sector is complex and dynamic, which makes the identification of threats and the assessment of vulnerabilities difficult tasks to achieve.

Nuclear Reactors, Materials, and Waste Sector: The Nuclear Reactors, Materials, and Waste Sector allows power reactors to provide electricity to the inhabitants of a country. The nuclear energy sector is, among the others,

one of the most safety-critical domains and well regulated to avoid harmful consequences to the users and the environment.

Transportation Systems Sector: The Transportation Systems wraps all the structures, both physical or virtual, people and means used to to move quickly, safely, and securely people and goods through places. In the U.S.A., seven subsectors belong to this field: Aviation, Highway and Motor Carrier, Maritime Transportation System, Mass Transit and Passenger Rail, Pipeline Systems, Freight Rail and Postal and Shipping.

Water and Wastewater Systems Sector: The Water and Wastewater Systems Sector is central to a nation public health sector, because safe drinking water is a prerequisite for protecting public health and all human activities. The Sector provides for public health and environmental protection. It is considered critical because a drinking water contamination accident or the denial of drinking water services would have a catastrophic impact across a Nation resulting in a large numbers of illnesses or casualties. Indeed, it is vulnerable to a variety of attacks, including contamination with deadly agents, physical attacks, such as the release of toxic chemicals and cyberattacks.

Each of these sectors has unique characteristics and operational modes. Thus, different risk profiles to characterize and mitigate. In the U.S.A., the Sector Specific Agencies (SSAs) were created to leverage expertise and institutional knowledge to enhance the protection and resilience of the national critical infrastructure. Each sector has a designated SSA, which employs its particular expertise to coordinate and collaborate with DHS and other relevant Federal departments and agencies for the protection of the infrastructure.

While the definitions may vary slightly depending the entities involved, Critical Infrastructures are generally considered as the key systems for the society. In all cases, they are composed of both physical (e.g., buildings, facilities, etc.) and virtual elements (e.g., data, networks, etc.). What constitutes the "critical" portion may vary as well. Other countries define different sectors from those reported above according to the available governement organizations. In any case, the low-level structures (e.g., hospital, waterduct, etc.) to protect remain the same. Typically, these might include elements of information and communications technologies (ICT), energy, public health, transportation and all the other sectors that a country determines too important in the sense they provide security to the nation inhabitants and support economic stability.

1.2 Critical Infrastructures Technologies

According to the Section 1.1, there are several terms and technologies associated to the concept of Critical Infrastructure.

In particular, this thesis focuses on the protection of Critical Information Infrastructures (CII). Independently on the sector, each Critical Infrastructure is composed of physical and virtual parts. Within the virtual parts, there are ICT systems that control, process, transmit or store information in any form, such as data, voice, etc. All this information can be considered vital to the functioning of the critical infrastructure. Also for CIIs, there are several possible definitions. In [70], CII is defined as any "ICT systems that are Critical Infrastructures for themselves or that are essential for the operation of Critical Infrastructures (telecommunications, computers/software, Internet, satellites, etc.)". The CIIs include all the systems and the services of a CI. The CIIs include also the communication (e.g., the telephone network, the Internet, the cabled and wireless networks and terrestrial and satellite communication networks). Moreover, CIIs might contain other infrastructures as well. The CIIs constitute the underlying support platform for all Critical Infrastructures. Also CIIs are at risk of attacks, disaster and, for their nature, technical failures. The high dependence of Critical Infrastructures on CIIs makes the last of vital importance.

Several types of systems, such as embedded, mobile, Cyber-Physical Systems (CPSs) and industrial control systems (ICSs) are used in critical infrastructures. One of the main components in a CII is an ICS, which includes one or more types of control systems that monitor processes and control flows of information. Operators must continuously monitor and control many different sections a Critical Infrastructure to ensure its correct operation. During the last decades this remote command and control has been made feasible due to the development of networking technologies together with ICS. ICSs are command and control networks and systems designed to support industrial processes [116].

A large and estabilished subcategory of ICSs are Supervisory Control and Data Acquisition (SCADA) systems. This technology allows for real-time data collection by means of sensors. Data are then transferred to a control and monitoring part of the system, across local communication links or through the Internet. The control and monitoring system is generally physically distant from the sensors.

Another technology parallel to SCADA, in past few years, are the so called Programmable Logic Controllers (PLCs). Also PLCs are important components belonging to the ICSs category. PLCs support manufacturing automation (e.g., production lines machinery) usually within the industrial field. PLCs are remote or distributed devices automating specific operations. The commands to operate are issued by qualified personnel or can be pre-programmed as set of actions.

Nowadays, both PLC and SCADA are types of technology that can be considered slightly outdated when compared with the Internet-of-Things (IoT) and the so-called Industry 4.0. The IoT is an important concept including a wide range of networked devices and sensors used to create a new set of applications. The IoT is deeply changing several fields by enabling the diffusion of home-automation

concepts, such as smart energy management and "smart-homes", or new medical devices revolutionizing health care sectors [214]. In the context of Critical Infrastructures, the integration of Industrial Internet-of-Things (IIoT) into the technologies employed is aiming to improve efficiency in many crucial areas [155].

The increasing flexibility of these technologies introduce several advantages, but also disadvantages. This exposes the Critical Infrastructures to cyber-risks and to faults, as will be discussed in Sect. 1.4 and in Sect. 1.3. The criticality of the infrastructure results in a demand for building such systems to be resilient and with protection against cyber-threats in mind, even more than conventional ICT systems, considering that a failing Critical Infrastructure could result in a cascading failure of systems, provoking also fatal effects [182].

1.3 Security considerations

Security is among the most important aspects of a Critical Infrastructure. Insufficient security measures or even absence of appropriate security mechanisms, may lead to the distruption of a Critical Infrastructure, turning into catastrophic consequences to physical or economic security, public health or safety of citizens of a nation.

The new technological advancements that are integrated into Critical Infrastructures lead to new cyber-security threats. The deriving risks are required to be taken into account beforehand and to be managed with specific security solutions. There are constant growing concerns about how to effectively protect Critical Infrastructures, given their crucial importance for the society and the environment they are embbedded in. Massive technological advances in this field for performance, efficiency and productivity increase the possible risks. Moreover, nowadays, these systems work no longer in isolation. Instead, they are becoming integrated into other systems shaping inter-dependence with other infrastructures [11, 14]. This requires that Critical Infrastructures must be kept secure from any possible source of danger, to avoid the negative consequences. Thus, it is important to address effectively potential security threats to successfully protect these systems [16]. Positive gains from new technologies, such as IIoT, are counterbalanced from improper security design foundations of new devices increasingly integrated in such systems. Many of these IoT devices address security as a remedy. Different vendors of the same device can follow different security recomendation, caused by the lack of an accepted security standards. A single device can represents a security weak point that an attacker can exploit to compromise a larger network. Thus, the lack of security standards among IoT devices is a major concern [155] because of the impact of a possible corruption on a Critical Infrastructure.

The ICSs employed in a Critical Infrastructure have some resemblance to common information systems. However, in the past, ICSs were traditionally isolated

ad-hoc systems with proprietary devices, software and control protocols [201]. This represents one of the major weaknesses in ICSs: communication protocols and their implementations can be one of the main sources of vulnerabilities, according to [38].

PLCs can be considered almost as any common desktop PCs, so they are vulnerable to the same types of attacks as traditional ICT systems [206, 156, 44]. However, in some cases, the security of SCADA systems can be more challenging than the traditional ICT systems. Indeed, early SCADA designs do not provide industrial systems with the protection against cyber attacks, because the majority of Critical Infrastructures were completely isolated. Old facilities employing these kinds of control systems are still in use as of today. However, they can be partially interconnected to other networks flowing to the Internet. This public surface is the source of attack for malicious and remote attackers, i.e., cyber terrorists. Although there exists protection countermeasures, such as firewalls, these might fail and in some cases they might not be even used because they were absent in the design of old control systems [39]. Moreover, common communication protocols are not suitable for all possibile control systems. Indeed, SCADA and other industrial protocols (e.g., Modbus/TCP and DNP3) are critical remote communications of data and control of the devices. However, these protocols were not designed considering the security aspect and, in some cases, they lack basic security primitives, such as authentication to execute commands [39].

Considering the technological advancements and the spreading of new techonlogical devices in CPSs, the number of cyber-attacks directed towards power stations, gas, and nuclear control systems is ever increasing [16]. In the past, the attacks performed were limited to Denial of Service (DoS) or Man-in-the-Middle (MITM) [167]. Instead, recent attacks are increasingly complex. The damage is aggravated in the case the attack targets a CI, given the repercussion to the community served by the system.

On December 23, 2015 a regional electricity distribution company in Ukraine reported service outages to customers. Approximately 225,000 users lost power across various areas. Shortly after the attack, Ukrainian government officially claimed that the outages were caused by a cyber attack. The attackers employed a variety of capabilities, including spear phishing emails and modification of documents by embedding malwares. The attackers were able to harvest credentials and sensitive information to gain access to the ICS network [52].

Another example of dangerous security attacks targeting the Critical Infrastructures is Stuxnet [140]. Stuxnet was reported in Natanz, Iran in June 2010. This complex attack has apparently infected over 60,000 computers, more than half of them in Iran. However, other countries affected included India, Indonesia, China, Azerbaijan, South Korea, Malaysia, the United States, the United Kingdom, Australia, Finland and Germany. Stuxnet is a sophisticated cyber attack designed to penetrate and establish control over remote systems almost autonomously [82]. The attack targeted only ICS controllers from one specific manufacturer (Siemens).

1.4 Safety domain

Besides security, also the safety aspects is important in the context of Critical Infrastructures. A safety problem in a subsytem of a Critical infrastructure may result in dangerous issues for a Nation. The CPSs and their components employed in the field of Critical Infrastructures differ from the traditional components for the operational system requirements. Indeed, during the design and the lifecycle of CPS, several aspects need to be considered, such as: components temperature range of operation, power supply, redundancy and fault tolerance properties, testing policies, etc. [238]. Human errors like misconfiguration or poor maintenance of the CPSs also negatively contribute to the system safety.

Safety aspects are tighly related to security. In both domains, adverse events might result in system corruption, which brings negative consequences. This is especially important for a subtype of systems, i.e., safety-critical systems. In these systems a failure could result in loss of lifes, significant property damage, or damage to the environment. There are several well-known examples of such systems, such as medical devices, flight control, nuclear power plants, etc. New technologies are often embedded in control systems, originating new failure modes to be taken into account during the design phase of a system in order to guarantee adequate safety mechanisms able to avoid, manage or mitigate the faults [134]. In this sense, safety and security are therefore two critical aspects for a CPS employed in a Critical Infrastructure, both sharing the same objective: protecting the CPS from risks due to accidental failures (safety) or due to intentional attacks (security).

1.5 Goal of the Thesis

The most ambitious objective of this thesis is to try to guarantee an improved security and resilience for the Critical Infrastructures.

To achieve this goal, I focused on the most important requirements that a Critical Infrastructure must possess, i.e., safety and security. In this context, a Critical Infrastructure is considered as a safety-critical system (whose failure could result in loss of human lifes or serious injuries, significant property damage or damage to the environment) and as a security-critical system (in which sensitive data must be protected against external malicious attackers or accidental disclosure).

Due to recent technological advancements, these systems largely exploit heterogeneous technologies. The heterogeneous components are used as computational devices to improve the performance or to provide greater flexibility of the whole system. In this work, heterogeneous system employed in critical infrastructures are analyzed under both safety and security domains.

More in details, the contributions of this thesis are as follows.

Chapter 2 presents an overview of the heterogeneous technologies employed in

CIs and in CPSs. The focus will be on the components largely used as accelerators or integrated to provide additional functionalities. In particular, are presented technologies exploiting reconfigurability, parallel computation and security-oriented features.

Chapter 3 offers an updated review of the State-of-the-Art, considering several works carried out in the domain of security. It explores the related works on the security of the microarchitecture of processors, which gained interest during recent years due to the new generation of side-channel attacks. In addition, an overview of security issues related to possible attacks on reconfigurable platforms is also provided.

Chapter 4 focuses on the security at the microarchitectural level. This chapter provides an analysis of a vulnerability exploiting a side-channel attack on a consumer microprocessor. It analyzes the interplay between two relevant design aspects of a critical system, i.e., safety and security. Moreover, this chapter presents the work published in: [46, 45].

In Chapter 5 the focus shifts on the protection of the communications against security attacks. In particular, the systems considered are mobile heterogeneous platforms, i.e., portable systems equipped with reconfigurable logic. Moreover, this chapter presents the work published in [47, 48].

Chapter 6 details the development of the open-source library for an heterogeneous device exposing functions typical of hardware security modules. It analyzes the security-oriented features of the firmware of the hardware device and the libraries of the host. An attack is performed against the platform to assess its robustness from the security point of view.

Finally Chapter 7 concludes the dissertation, summarizing the key aspects related for the protection of Critical Infrastructures.

Chapter 2

Heterogeneous Computing Architectures

The computing platform employed in Critical Infrastructures and corresponding Cyber-Physical Systems are based on several types of heterogeneous components. In addition to microprocessors, each platform can contain many types of components, such as sensors, peripherals, memories, accelerators, etc. This chapter presents an overview of the computing technologies employed in this kind of systems.

2.1 Heterogeneity in CPS

Critical Infrastructures are composed by various types of systems, such as Cyber-Physical Systems. CPSs are made up of physical parts, virtual parts and cyber-physical parts. The cyber components are employed to monitor and control processes in the real world, and receive data sensed from the external environment. This provides a feedback loop to help dynamically adjust the behavior where physical and cyber overlap.

As stated in [151], CPSs can be considered as the product resulting from the integration of heterogeneous systems. They are composed of heterogeneous parts and distributed systems that integrate and interact with other information systems and physical systems. For example, CPSs employed in existing manufacturing practices are blended into the production, logistics and services [144].

Recent technological advancements have led to greater sensor availability and accessibility, high-throughput data processing systems and computer networks. This forced a growing number companies and factories of the industrial environment to shift towards the adoption of different technologies. In consideration of the demands on the needed functionalities, the safety, the security and the cost of a CPS, the choice of implementation involves the use of a variety of different types of technologies in both architecture and functional domains, originating various

solutions. In recent years, the range of available alternatives has transformed to include different types of devices. The heterogeneity is represented in the design of the various features and in its implementation, which includes different components such as Digital Signal Processors (DSPs), microprocessors, memories, components with networking capabilities, sensors, actuators, etc. All these components are often integrated into a single chip. Moreover, in these platforms one or more types of programmable components is integrated on the same system to offer specialized capabilities giving the possibility to handle particular tasks. As an example, a programmable logic device can be combined with a microprocessor, resulting in an heterogeneous system.

The following sections, illustrates the various types of components that can be integrated among the computing components of a CPSs. This integration produces some form of gain for the whole systems. The benefits are usually related to the different system design dimensions, such as:

- Performances: the presence of hardware accelerators ad-hoc engineered for a specific application increases the performances;
- Power Consumption: the employment of low-power devices for specific tasks decreases the overall power consumption;
- Flexibility: the capability to modify or improve the system setup leads to maximize a certain cost dimension (not necessarily fixed);
- Security: certain components can bring improvements and specific functionalities from a security point of view.

However, these dimensions are inter-related. For example, a component that brings a speed-up in performances might increase as well the total power consumption. Thus, it is necessary to carefully design the system resorting to effective design space exploration techniques (e.g., [202]).

2.2 CPU

Traditionally, the cyber part of CPS and the backbone of the CII rely on CPUs as the main components to build systems and networks. CPUs are an established technology that dates back around 1950 and they are at the base of every type of computer. Processors are able to carry out computations and possess general-purpose computational capabilities, i.e., they are not optimized for a specific domain. The trend in processor development has been toward the integration of multiple processing units (or computational cores) in the same CPU. Currently, desktop CPUs are equipped with 8 cores, while CPUs used in server computers have 16 cores available. In this way, CPUs become the optimal choice for handling

concurrent processes. Indeed the instructions to be executed can be run on separate cores of the same CPU at the same time, increasing overall computational performance.

Although software applications support techniques such as multithreading or other parallel computing techniques, the multi-core CPU architecture might not be capable enough to handle large-scale parallel computations. CPUs can exploit multithread processing to achieve faster execution. However, in case of very high number of threads, the performance on CPU do not scale well. Indeed, the context switching operation causes a not negligible overhead resulting in performance penalties. Also the dispatcher, the scheduler and the cache operations do not scale well when the number of threads is high.

2.3 GPU

GPU (Graphics Processing Unit) is a relatively new technology designed for parallelizable problems. It was initially created specifically to handle graphics-intensive applications, however it has become more capable of general computations (i.e., so-called General-purpose computing on Graphics Processing Units - GPGPU). A GPU is a heterogeneous chip multi-processor.

From the architecture point of view, a GPU is composed of hundreds of processing units. The high number of cores allows to handle thousands of threads simultaneously. Conversely, multi-core CPUs are composed by few cores and are able to handle quite few software threads. However, CPUs are designed to minimize latency resorting to cache memories. Instead, GPUs are high-latency but high-throughput processors. This consideration allows for GPUs to have a better transistor/area ratio to be employed for more ALUs and therefore to be able to run many more threads of computation. A simplified comparison between the architecure of CPU and GPU is shown in Fig. 2.1.

Running tens of thousands of computational threads, lead to synchronizations issues. To avoid synchornization stalls and preserve the speedup, each computational thread is required to be independent from each other. GPU is indeed an architecture in which multiple processing units process multiple data streams in parallel, i.e., Single Instruction stream, Multiple Data stream (SIMD) architecture. SIMD represents a category of instructions in which perform the same operation on multiple registers simultaneously. The data parallel problems are those that benefit the most from the SIMD architecture. Exploiting data parallelism allows to distributing the data across different processing units. All of them perform in parallel the same operation, but on a different small piece of data. Several problems belong to this category, such as scientific computing, physics, simulations and especially graphics, image and video processing. CPUs also have SIMD instructions. However, a CPU has a limited number of sequential cores, while a GPU employs thousands

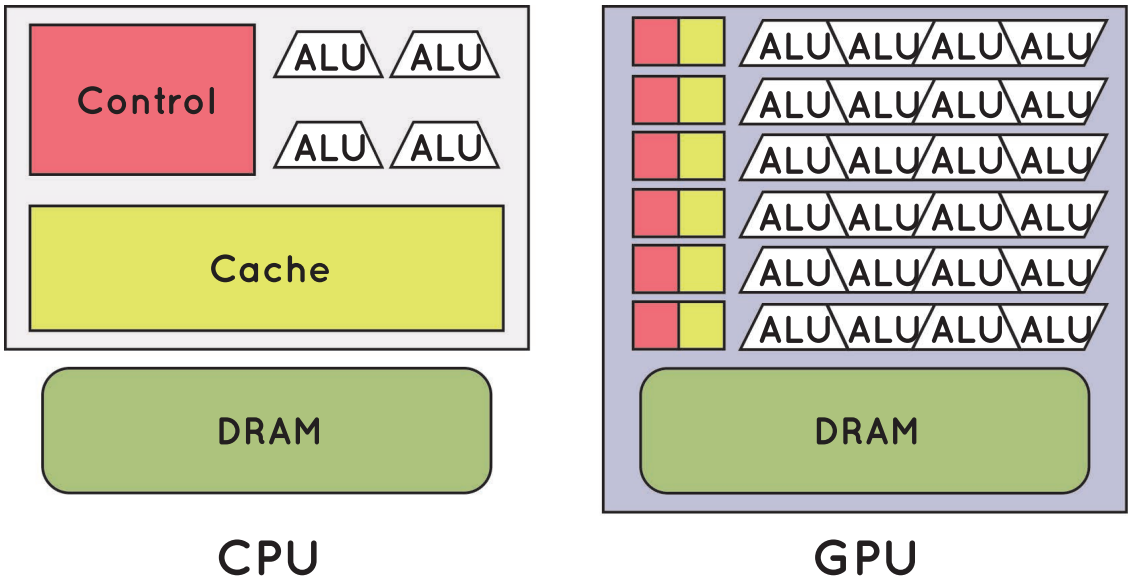

Figure 2.1: Comparison of simplified GPU and CPU architectures.

of parallel cores. A comparison of the main distiguishing features between CPUs and GPUs is reported in Table 2.1.

Table 2.1: Comparison of the main features of GPU and CPU.

Feature	**GPU**	**CPU**
Parallelism	Data	Task
Instruction	Same instruction, different data	Different instruction, different data
Latency	High	Low
Throughput	Moderate	High
Cache size	Small	Large
No. of Cores	Hundreds	Tens
No. of Threads	Tens of thousand	Thousand
Thread Management	Implicit	Explicit
Power Consumption	High	Low to High

GPUs use various optimizations to improve throughput. Usually the GPU chip is embedded into a card, which is attached to the host platform where the CPU resides. Data transfer between the CPU and the GPU is time-consuming. Thus, GPUs often integrate memories, such as on-chip memory and local caches, to reduce bandwidth to external memory. To improve the throughput, GPUs use stream processing where the same series of operations, called *kernels functions*, are executed against each element composing a set of data, that is the *stream*. These kernel

functions are usually pipelined to avoid stalls and performance loss. Moreover, to avoid performance losses deriving by thousand of thread, their management occurs in hardware by the GPU itself.

2.4 CPU + GPU

An architecture composed by a CPU coupled with a GPU as an external computational device can suffer of different factors that limit the efficacy brought by the heterogeneity of the components.

One of these limitations derive from the communication between the CPU and the GPU. From the architectural point of view, the data to be processed reside in the main memory. In order to exploit the parallel computational capabilities of the GPUs, the data have to be moved from the main memory to the GPU memory. After the kernel functions have been executed on the GPUs, the output of the computation has to be moved back to the main memory. This kind of data transfers requires time, which is related to the available interconnection bandwidth. Altough the transfers unburden the CPU from computation, they constitute a bottleneck. Thus, this slow down partially defeats the benefits of the GPU computational efficiency and greatly reduce the performance of the whole sytem.

Another issue related to GPUs is the power consumption. Given the high number of computational resources, a GPUs is considerably more power-hungry with respect to a CPU.

For these reasons, technology trends show that CPUs vendors are shifting towards a tighter integration of CPU and GPU on the same die. Already back in 2010, AMD released Fusion APUs, Intel released Sandy Bridge architecture and ARM with MALI represented technological solutions that integrate general purpose CPUs together with GPUs.

These heterogeneous devices merge the functionalities of both architectures, by combining the benefits and limiting the drawbacks of the two separate components. The integration of CPU and GPU into a single unit on the same chip provides several advantages. The components integration allows the use of shared structures which translates in a reduction of costs. Moreover, CPU+GPU architecture reduces the bottleneck of communication because no explicit data transfers are required between the CPU and GPU. Finally, integrating the GPU and CPU has a positive impact in power consumption than having a CPU with a separate and dedicated GPU. Altough currently these integrated devices are quite powerful, a modern dedicated GPU almost always outweighs the CPU+GPU from the computational point of view.

2.5 FPGA

Field-Programmable Gate Array (FPGA) is a type of reconfigurable hardware. It is essentially a semiconductor device as an integrated circuit (IC), but it offers the possibility to be re-programmed, i.e., it can change the functionality of the hardware itself. The programming can be performed once or multiple times. One of the main advantages of FPGAs is that the circuit can be programmed in-the-field, i.e., after the manufacturing process.

This characteristic allows the user to redesign repeatedly an hardware solution to solve specific types of problems more efficiently and to deploy it on the same device without requiring expensive procedures. The design can also be changed after the final product embedding an FPGAs has been shipped to customers out in the field. This reconfigurable feature also distinguishes FPGAs from Application Specific Integrated Circuits (ASICs), which are custom IC designed for specific tasks that cannot be modified after manufacturing. Moreover, the manufacturing time for ASIC might require many months. In addition to their flexibility, limited product development costs and continuously growing computing capabilities make FPGAs a viable alternative to ASICs. Modern FPGAs offer programmable logic blocks, flexible clock generation and interconnection circuitry, memory blocks and embedded hard DSPs. Some specific device families also feature special embedded hard blocks, such as entire microprocessors, memory controllers, RAM blocks transceiver I/O providing high-speed, allowing additional flexibility and incresing computational capabilities. Other families are characterized by low-power consumption. On some types of applications FPGAs can greatly improve the performance over conventional microprocessors. The reconfigurable hardware can be used to implement application specific accelerators on demand. To improve the performances the accelarators can exploits *parallelism* property, where multiple independent operations are executed at the same time in parallel. This is very crucial in applications that performs operations such as convolution, bit manipulation, multiplication, etc. In these cases, FPGAs can execute these instructions on multiple data at once, with low control overhead and reducing the power consumption, unlike conventional microprocessors. For these reasons, FPGA devices can be exploited in a variety of applications ranging from HPC (High Performance Computing) and enterprise environments to mobile devices. FPGA can be employed in critical application scenarios to increase system dependability and lifetime [66, 50, 64]. They can be employed as a stand-alone platform (i.e., like a normal IC) or can be integrated in heterogeneous computing architectures of reconfigurable systems. FPGAs can be paired with a CPU, composing a system in which hardware acceleration and processing units run in parallel, effectively exploiting all the components and enhancing the throughput of the system. Among the reasons behind FPGAs popularity, there is a simplified design flow with respect to ASIC ICs. As a consequence, there is a reduction of design costs which translates to a shorter Time-To-Market (TTM)

leading to revenues increase.

The architecture of an FPGA is made up of a regular structure composed of a two-dimensional array of Configurable Logic Blocks (CLBs). The elements in a logic block can be configured to implement a certain set of functions. Each logic block has a fixed number of inputs and outputs. The CLB can contain Look-Up Tables (LUTs), Flip-Flops (FFs), and multiplexers. However, FPGAs realized from different vendors use different architectures and also different components. The basic structure of a LUT-based logic block[1], shown in Fig. 2.2, is composed of:

- LUT: logic device which gives in output a specified value as a function of its input;
- Flip-Flop (D-FF): asynchronous set and clear flip-flop, it is used as storage elements for sequential circuits;
- Mux: multiplexer used to bypass the D-FF in the case of pure combinatory cells.

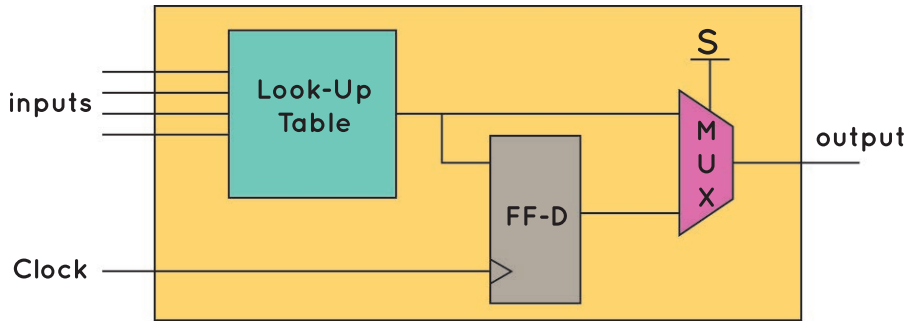

Figure 2.2: LUT-based FPGA Logic Block (S is set from the configuration bitstream).

The array of logic blocks is entangled within an interconnection network, i.e., each reconfigurable block is surrounded with communication lines. Also the interconnection network can be configured through Programmable Switch Matrices (PSMs). Modern FPGA devices usually embed additional special purpose blocks, such as I/O Blocks, RAM Blocks, Multipliers, CPUs, and dedicated DSPs. Figure 2.3 shows the high-level internal architecture of a generic FPGA.

The programmable components in an FPGA must first be configured, in order to implement the user-defined design. To configure the FPGA one need to set the logic blocks and program accordingly the interconnections. The *bitstream* describes the developed design in terms of boolean functions and interconnections to be set

[1]There could be also Mux- or ALU-based logic blocks

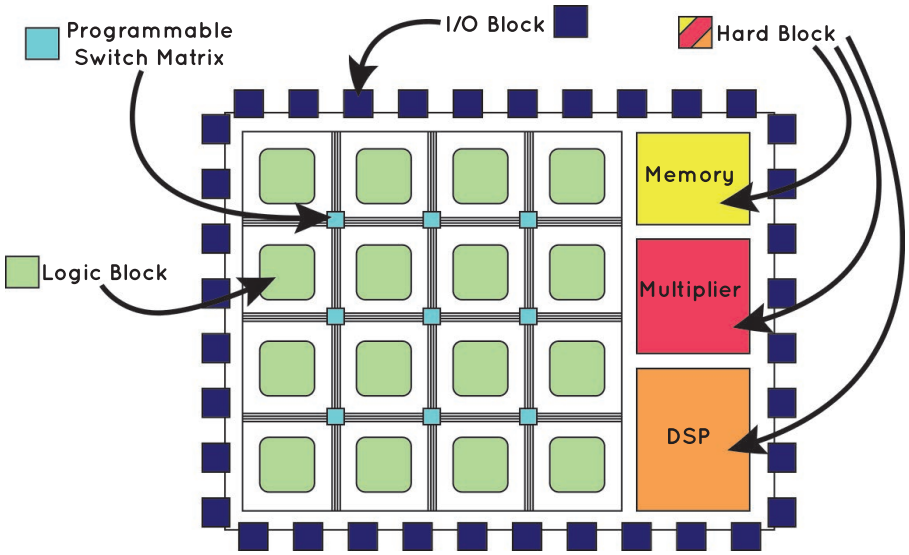

Figure 2.3: FPGA internal architecture.

up on the FPGA. The bitstream is the final result of the design phase and it is generated by the tools aiding the design provided by the vendors of the FPGA. The configuration process, described visually in Fig. 2.4 is essentially the deployment of the design. Basically, it involves the transfer of the bitstream into the target device.

The bitstream flows from the configuration memory to the Configuration Layer of the FPGA. This layer determines the kind of computation that shall be performed. This level changes the behavior of the above Logic Layer, which is the one actually performing the computations.

FPGAs can be distinguished based on the technology employed in their configuration memory:

- PROM-based: the memory is writeable only once. It implies that the FPGA cannot be reprogrammed, but it is One-Time-Programmable (OTP);
- (E)EPROM/Flash-based: the memory is non-volatile. However this technology allows limited re-writeability;
- SRAM based: exploit static memory cells. It is a volatile memory, thus FPGA requires to be reconfigure every time after the boot up.
- Hybrid: the memory is composed by EEPROM and SRAM memories. In this case, the power consumption is generally higher and is required a larger area.

Apart from the reconfigurability feature, some families of FPGAs also provide adaptability through a feature called Dynamic Partial Reconfiguration (DPR). DPR

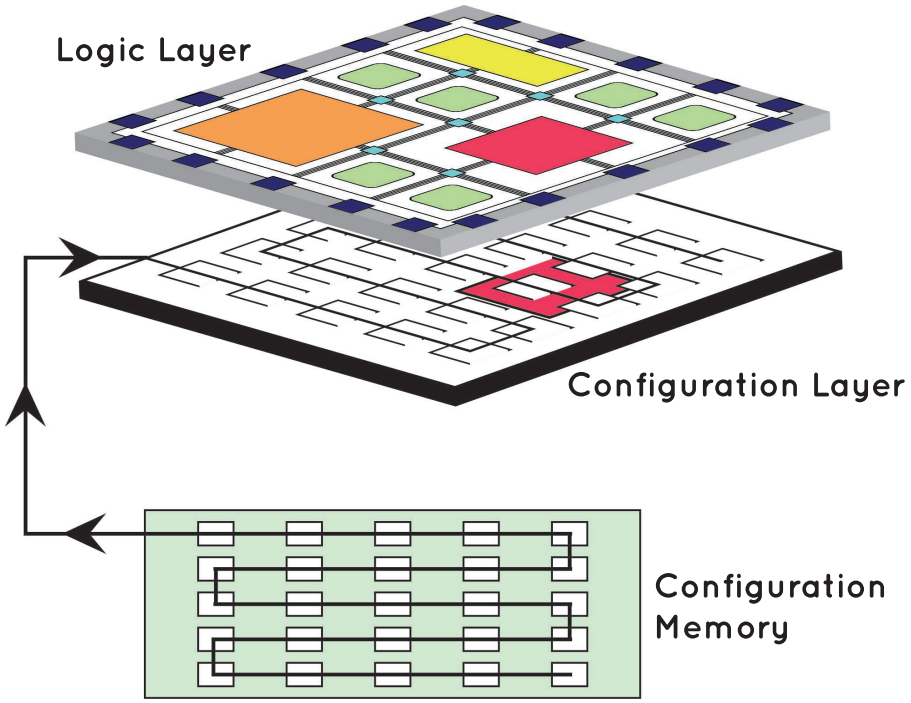

Figure 2.4: FPGA Configuration process.

provides the opportunity to alter at run-time the functionalities impemented in selected parts of the FPGA, while the remaining portion of the system stays fully operational, without interruptions. This feature enhances the native flexibility of FPGAs by enabling the capabilities for time-multiplexing of hardware resources. DPR can be exploited in many application fields where it is necessary to optimize designs [147] and to fulfill severe area constraints [169], to create a system-on-a-chip with a high degree of flexibility [252], to realize adaptive algorithms [177], to be more resilient against errors [170], etc.

2.6 Hardware Security Modules

Modern general-purpose computers, such as personal computers, smartphones, tablets, etc. usually run many applications concurrently in a shared environment. The Operating System (OS) does not guarantee that the resources used by various applications are entirely separated, because this would hinder usability. This exposes sensitive data used by applications to great risk.

Altough all those applications that perform critical operations on sensitive data might resort to some encryption mechanism to protect the information, a malicious application might be able to gain access to the cryptographic keys stored in RAM or in permanent storage.

Using dedicated hardware specifically to enhance the security of a system is often advisable, especially in contexts where security is a key aspect. In the real world, this occurs very frequently: for example, banks offer various types of smart cards to their clients for authorization of transactions, and also mobile network providers have been using smart cards to authenticate their subscribers for accessing their services. Smart cards, however, do not cover all the potential scenario for cryptography: very often, a more comprehensive and more flexible solution is needed.

Hardware Security Modules (HSMs) are special-purpose computational devices, tailored to provide a wide range of security features. HSMs can be used together with a general-purpose computer or integrated into a larger system. Simpler hardware tokens, such as smart cards and SIM cards, can be considered as a common example of HSM employed for authentication purposes. However, HSMs are able to perform additional tasks.

These devices can secure various sensitive data processing operations and provide robust mechanisms for authentication and encryption processes. They are usually anti-tamper hardware devices realizing security and cryptographic functions. HSMs can perform several cryptographic algorithms, without exposing the secret keys employed. Most of them can aid the evironment where they are employed to manage multiple keys associated with respective processes. In general, organizations make extensive use of HSMs to enforce their security policies and efficiently maintain their access controls mechanisms. Basically, security modules are used for encryption/decryption of various cipher algorithms and key management operations (e.g., key generation, key storing, key exchange).

Appropriate handling of cryptographic secrets for the practical use of cryptography is crucial. A cryptographic key goes through several life stages such as key generation, secure storage, secure key exchange and distribution, backup and finally destruction. An HSM is specifically used to protect these secret information in every stages of their life cycle. HSMs control logical and physical security of cryptographic keys from attackers and unauthorized activity.

HSMs can also provide improved cryptographic performance. Its integration in a security-critical environment results in a more effective architecture. In this case, the HSM is an advantageous component to provide security functionalities able to reduce security risks and to achieves high performance in cryptographic operations, by offloading expensive computations from the central unit.

The functionalities and capababilites of HSMs are ever changing and increasing. An extended and detailed list of standardized characteristics is provided in the PCI PTS HSM v3 [183]. Considering the importance of HSMs, it is essential that they provide some assurance of their security. FIPS 140-2 [84] is an international security standard used to approve cryptographic modules. In [84], four levels of security (Level 1 to Level 4), are defined: HSMs that are compliant with FIPS 140-2 security Level 3 and above provide the highest level of security.

Several commercial HSMs are available. Portable versions are available too. Common and wide-spread HSMs of this kind are reported below:

- YubiKey[2]: it is a USB device providing secure authentication. It provides one-time-password (OTP) and other authentication features. It is not programmable.
- USB Armory[3]: it is flash drive-sized computer in an USB device. It is open hardware and software. It includes a 800MHz ARM Cortex-A8 CPU, 512MB RAM and microSD card slot for storage. It can run some Linux distributions. It is programmable and customizable.
- SEcube™ [4]: it is an hardware chip. It is composed of a microcontroller, an FPGA and a smart card in a single package. It is open source (further details on this platform will be discussed in Chapter 6).
- TREZOR Bitcoin safe[5]: a hardware bitcoin wallet used to protect cryptocurrency funds. However it provides also password manager and secure authentication capabilities. It is not programmable.
- NitroKey[6]: it is a USB device providing secure authentication and encryption services. It is both hardware and software open-source.

Some of these devices are fully programmable, by being open-source. Those with closed-source code can no longer be independently reviewed for security flaws. There are devices also more similar to a normal computer, i.e., running an Operating System. Others are considered as *security tokens*, because they share some similarities with handheld devices. Usually, thay are shaped in a "key" form-factor and communicate with an host device through USB CDC device emulation. Some devices also act as a password manager, supporting key management operations. In general, this kind of devices can be also integrated into larger systems.

[2] https://www.yubico.com/

[3] https://inversepath.com/usbarmory.html

[4] https://www.secube.eu/

[5] https://trezor.io/

[6] https://www.nitrokey.com/

Chapter 3

Security Attacks

This chapter provides an updated review of the state-of-the-art regarding related works on security attacks with a focus on hardware-level attacks. First a global overview of the security landscape is provided, focusing on side-channel attacks. Then, microarchitectural security attacks are analyzed. Finally, a review on security issues regarding reconfigurable logic is described.

3.1 Security overview

(Cyber)Security is not a clearly demarcated field of academic study that lends itself readily to scientific investigation. Rather, it combines a multiplicity of disciplines, such as technical as well as, in some cases, behavioural [80].

According to NIST [132], computer security is defined as:

Computer Security *Measures and controls that ensure confidentiality, integrity, and availability of information system assets including hardware, software, firmware, and information being processed, stored, and communicated.*

Security has its roots in three pillars, i.e., confidentiality, integrity and availability, constituting the so-called **CIA triad**. These three properties establish the *key objectives* of the computer security. The interpretations of these three aspects vary, as do the contexts in which they arise. The interpretation of an aspect in a given environment is dictated by the needs of the individuals, customs, and laws of the particular organization [28].

Confidentiality is the property concerning the obfuscation of sensitive information or resources. This term involves two other concepts: (1) Data confidentiality: it concerns the protection of data that is private or confidential information by making it unavailable or undisclosed to unauthorized parties. (2) Privacy: Ensures that the information related to individuals can be controlled in terms of who can collect them and to whom they can be disclosed to. It is focused on individuals, such as persons, organizations or any other entity, especially on their sensitive data.

The need to keep information confidential stems from the diffused use of computers and computational devices in organizations handling classified information, including governments and industries. As an example, let us consider institutions such as military and civilian agencies. These institutions very often limit the access to information to those who need that information. Confidentiality enforces the so-called *"need to know"* principle which is a general principle, developed in the context of information security and in the management of security systems, according to which *only* the subjects who must perform information processing activities are authorized to process *only* the data essential to the fulfilment of the task assigned to them and the data must not be shared, communicated or sent to third parties who do not need it. This principle also applies to industrial companies, which desire to maintain their design and projects confidential and secure them from competitors that might try to steal and copy the property, allowing financial losses to arise. Further examples are given by every type of institutions that keep employee and personnel records secret.

Confidentiality is ensured through mechanisms of access control that regulate data access. *Access control mechanisms* allow authorized parties to access the protected data and deny access to unauthorized parties, thus preventing confidentiality breaches. One access control mechanism for preserving confidentiality is cryptography. Simplifying, cryptography transforms human-readable data through a process (i.e., encryption) so that the data becomes unrecognizable and incomprehensible. Encrypted data can only be accessed by individuals who possess the key to decrypt the data. However, the cryptographic key itself becomes another piece of information to be protected and to be kept secret.

Confidentiality also applies to the *existence of data*. The existence of data, in some cases, reveals more information than the data itself. Access control mechanisms can be used to disguises the existence of data, because the presence of data itself might reveal information that should be protected.

A further important component of confidentiality is *resources hiding*. Sometimes, companies try to mask their network configuration, as well as the technologies they use. Organizations may not allow everyone to know about the particular equipment they employ. Access control mechanisms provide these capabilities as well.

Integrity refers to the reliability of data or resources, and it is generally meant to prevent illegal, unwanted or unauthorized changes. This term covers two related concepts: (1) Data integrity: Ensures that information stored or exchanged is only changed in a defined and authorized way; (2) System integrity: Ensures that a system performs its intended function without any impairments and without unwanted or unauthorized modifications of the system behavior.

Integrity involves *data integrity*, which concerns the content of the information, and *origin integrity* which is related to the source of information.

Mechanisms to assure integrity can be divided in two classes: prevention mechanisms and detection mechanisms. On the one hand, prevention mechanisms aim to protect data integrity by denying any unauthorized attempts to modify the data or efforts to modify the data in unauthorized manner. Both attempts concern authorization. In the former attempts, only authorized users are allowed to make changes to data. The latter instead, occurs when an user properly authorized to make specific changes, tries to alter the data in an illegal way.

On the other hand, detection mechanisms do not seek to avoid nor to prevent violations of integrity. These mechanisms are simply used to report that the data can no longer be trusted. Detection mechanisms can evaluate events, originating from an user or from a system, to detect any issue concerning the integrity. They might as well analyze the data itself to recognize unwanted changes. Integrity is very different from confidentiality. With confidentiality, the information is either compromised or it is not, while integrity involves both correctness and trustworthiness of the data. This implies that the integrity of the data can be affected or compromised in every step, spanning from the source to the receiver of the data.

The availability ensures that a system operates efficiently and authorized users can take advantage of the services offered by the system.

Availability refers to the ability to use information or resources. Availability is an important aspect of reliability as well as of system design because an unavailable system does not offer any service and is then equivalent to having no system at all. From the security point of view, availability becomes relevant when someone might voluntarily tamper with the system by denying access to the services or to the data stored in the system itself. In this case, the system is unavailable and its users cannot be served. Usually this aspect is considered in the system design phase. At this stage, the designers hypothesize statistical models to analyze expected usage patterns and then decide the proper mechanism to ensure availability of the system and the continuity of the services. An attacker might force the system to work outside the area of the assumption model, so that the mechanisms for keeping the resources or data available are working in an environment for which they were not designed. As a consequence, the system is in an unpredictable behavior, which usually leads to failures. Identifying the intentional attempts to undermine availability of a systems, which are attacks called Denial-Of-Service (DoS), is a challenging problem. The difficulty lies in the pattern of access to the system or to its services, i.e., it is necessary to recognize those irregular access patterns traceable to intentional resource or environment manipulation.

The security field is huge. It is therefore important to understand *where* it is necessary to act within the security field. Several possible ways to organize the knowledge in this field have been proposed through taxonomies. A Taxonomy is defined as *"a system for naming and organizing things, especially plants and animals, into groups that share similar qualities"* [228]. However, it has to be considered that there is never one uniquely valid taxonomy in a specific domain,

but instead a taxonomy might be more representative and expressive than another one in a specific context. Several general taxonomies for security do exist [133, 153].

Scientific study is further complicated by the rapidly evolving nature of threats, the difficulty to undertake controlled experiments and the pace of technical change and innovation. Securing critical infrastructures and cyber-physical systems remains a challenging problem because the exponential growth of cyber-physical systems is accompanied with the rise of new security challenges. New vulnerabilities, threats and attacks have been introduced with technology advancements. As an example, the trend of using the Cloud will soon reach the Critical Infrastructures [182]. This trend also opens security concerns because existing methodologies and strategies for designing secure applications only tackle security issues and protection problems in either the Critical Infrastructures or the Cloud. Specific methods, techniques, tools or guidelines may support only a limited set of common Critical Infrastructure requirements [182]. Also SCADA systems are increasingly dependent on the Internet as provided by the carriers [89].

A systematic model to address the security concerns in the context of CPSs is missing [112]. This lack, is due to:

- diversity of CPSs: each CPSs is, in general, different from another. The differences can reside in the field of application, in the system architecture, in the interconnections, etc. Eventually, any sectors of critical infrastructures generates different security problems, presenting different risks and threats and thus techniques to address the security issues.

- heterogeneity of CPS components: different types of components are employed to compose a CPS or a part of it. Each one introduces security problems that require tailored techniques for securing a specific subsystem or component.

In particular, these reasons make difficult to study and address the security problems with just a unique generalized model. Recently, however, a general approach to model the security of a CPS has been shown in the survey [112].

Very often each application of technology, origins its own security issues. In the available literature, to tackle the security issues several works have been published. Each one addresses various security attacks and security models for a specific field of applications. In field of Telecommunication, the works [87, 215] offer an overview of the security challenges in Wireless Sensors Networks (WSNs). Related to the world of Automation, the survey of [68] presents an update review of attacks in Industrial CPSs, while the publication [17] shows security problems of IoT systems, more specifically in a Smart-home environment. In the Transportation sector, the work of [232] addresses the security in Autonomous Vehicles. In the area of Energy and Power systems, the authors of [225] present a solution for network security situation

awareness based on the power monitoring system network security metadata. In the recent work [59] in the field of Electricity Grid, the authors show a comparison of the benefits among attack models in a CPS belonging to this field. In [184] an up-to-date and comprehensive security analysis of Smart Grids framework is presented, as well as attack scenarios, attack detection and attack protection methods, estimation and control strategies from both communication and control perspectives. Finally, in the Healthcare sector, various security attacks affecting electronic healthcare systems are provided in [191].

A class of attacks that might lead to important security breaches are Side-Channel Attacks (SCAs). These attacks might be performed even if normal security and safety mechanism are correctly integrated. They might target devices, such as embedded systems, left unattended because physically located at the end nodes of a critical infrastructure (e.g., sensors and/or actuators, PLCs, networking devices, etc.), so that the attacker can have physical access to these devices. To worsen the criticality, some SCAs can take place also remotely.

3.2 Side-Channel Attacks

Ideally, the mechanisms to achieve information security are secure (or they are typically considered secure enough) in theory. However, a specific practical implementation of such methods might not be inherently secure. A possible implementation of a process might release other information as unintended output. Side-Channel Attacks (SCAs) are a category of security attacks exploiting specifically this type of information, which is *leaked* from the implementation of an algorithm rather than from the algorithm itself. This "side" information is sensed from a channel that lies in the environment, rather than being extracted directly from the target. In Fig. 3.1 is depicted the "side" information deriving from the implementation a process.

As an example, one can consider a software implementation of an encryption algorithm. The software application requires a certain time to run an encryption on a specific amount of data, e.g., proportional to the size of the data. Supposing to encrypt two chunks of data having different size, the difference in the execution time between the two encryptions might disclose information about the size of the chunks of encrypted data. Although the obtained information is not critical from the security perspective, this trivial example represents a good illustration of side-channel information, given that the time information obtained is completely unintended and produces, as a *side effect*, the information about data size. Moreover, in some cases, the information gathered might be used for other purposes.

Further examples of side-channel attacks might employ information gathered from electromagnetic emissions, power consumption, temperature, timing, glitch events, etc.

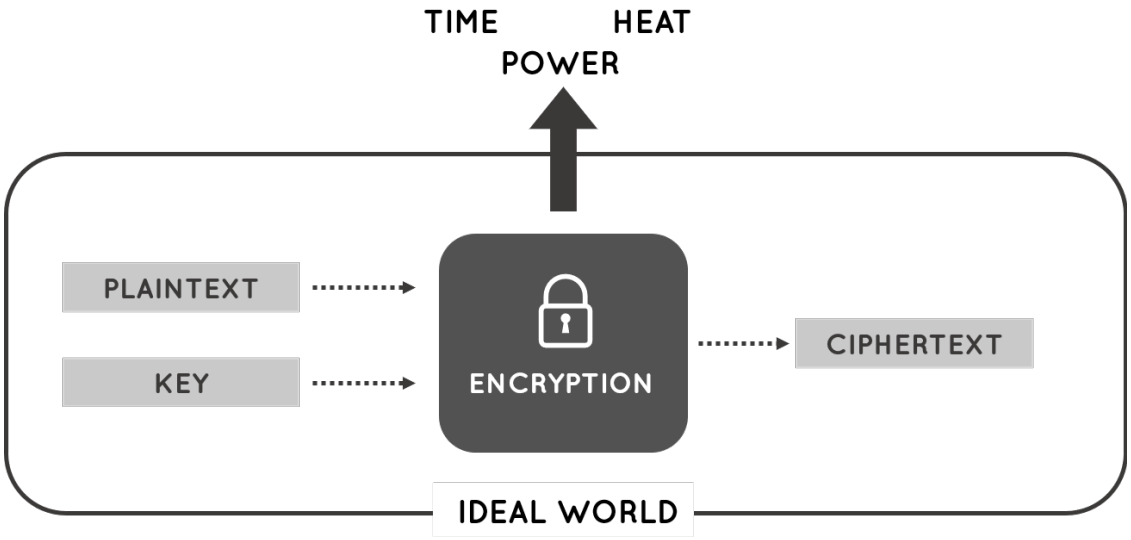

Figure 3.1: Side-Channel information leaked from an implementation of a process activity.

The early works on SCAs have been performed in 1996 by P. Kocher in [138]. This groundbreaking research work was an attack exploiting variations in an algorithm's execution time. This pioneering work, together with another [135], marked the start of a new research field on side-channel attacks. Years later, side-channel attacks were performed employing essentially any observable environmental change (e.g., electromagnetic radiation, acoustic emissions, etc.) induced by different kinds of computations. SCAs have been also performed on specific target devices, such as smart cards [194, 154], because the hardware could leak components of its internal state including potentially sensitive information via behavioral and timing differences.

This class of attacks is subtle to detect and might be hard to counteract, indeed already in 1985 the U.S. Department of Defense was specifically concerned about SCAs in its so-called *Orange Book* [141].

3.2.1 Side-Channel Attacks - Taxonomy

Several SCAs are possible. There exists multiple taxonomy as well [218, 29, 139]. Fig. 3.2 depicts a simplified taxonomy representing the various kinds of Side-Channel Attacks.

SCAs can be classified according to two orthogonal dimensions, i.e., based on the capabilities of the attacker and on the invasivity of the attack itself.

Considering the attacker, a SCA can be classified as:

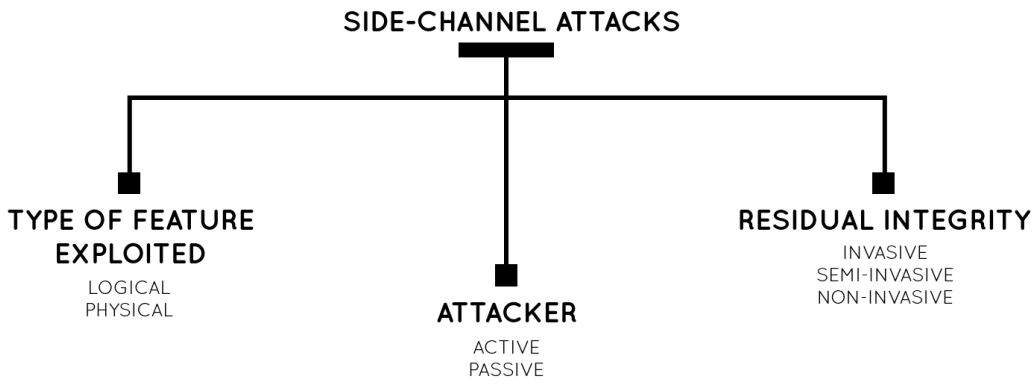

Figure 3.2: Taxonomy of Side-Channel Attacks.

- Active SCA: where an attacker purposely interferes with the devices by modifying or influencing it through a side channel, e.g., by means of an external interface or environmental conditions. The attacker thus controls the device's operations in a manner that leads to unexpected behaviors, which enables for possible attacks.
- Passive SCA: where an attacker is limited to observe the side-channel without interacting with its target. However, the target during operation influences a side channel potentially leaking sensitive information.

Fig. 3.3 shows a graphical representation of the difference between an active and a passive SCA.

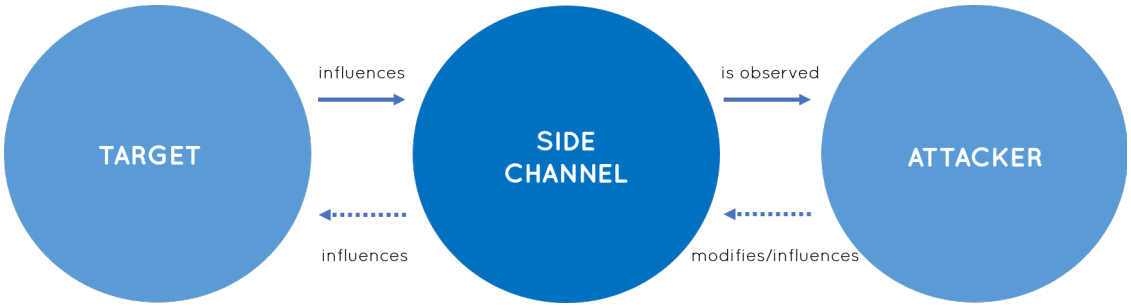

Figure 3.3: Representation of active vs passive SCA.

The other dimension is related to the invasivity of the attack and considers the integrity of the attack target after the attack is performed. In this case, the SCA is divided into:

- Invasive: performing the attack requires interactions with the target resulting in a destructive impact.

- Semi-Invasive: after the attack, the target presents minor impacts.
- Non-Invasive: the attack target is left untouched.

An additional characteristic of this type of attacks, is the employed channel of information leakage. The first works on SCAs of P. Kocher exploited timing information [138] and power consumption [135] as physical measured quantities. Others works classified as non-invasive SCAs, employed electromagnetic analysis [194], simple power analysis (SPA) [158], temperature information [115], glitches forced in the clock signal [193] or glitches due to power [91]. The semi-invasive attacks, according to [139], are performed exploiting faults attacks. The information retrieved are interpreted resorting to faults analysis [27, 146]. In this case, the attacker manipulates the environment in a way that influences the attack target. The target is thus forced to work outside the defined area for which it was originally designed in order to intentionally trigger faults, such as faulty outputs or fault behavior. Fault injections are also used to obtain special information leakage under the faulty environment. Among the semi-invasive attacks, according to [139], laser fault injection [200], electromagnetic fault injection [176], and optical fault injection [245, 216]. To be carried out, faults attacks often require a certain degree of physical access to the attack target. Finally, invasive attacks involve physical alterations provoking total or partial destruction of the attack target. If the target is a chip or a device, a possible invasive SCA consists into the removal of external packaging to get direct access to the internal components. After the removal of the enclosure, the attacker can locate a data bus and physically analyze through probing needles sensitive data transfers.

Although the majority of the SCAs are hardware-based, there exists also software-based microarchitectural attacks [143]. Also this type of side channel attacks take advantage of indirect flows of data to extract sensitive information, but the flow resides within microprocessors. Thus, as discussed in the two surveys [218, 139], a further classification can be according to the type of information exploited, i.e., either logical or physical properties. Attackers resorting to this type of SCA are able to execute some software on the attack target in order to exploit both physical properties as well as software features (logical properties, such as the memory footprint) [218]. Details about this class of SCAs are further elaborated in Sect. 3.3 and in Sect. 3.3.1.

3.3 Microarchitectural Side-Channel Attacks

The components employed in CPSs and in CIs operate under a security critical framework. In general, security and safety countermeasures are employed in these systems to protect from well-know cyber attacks, such as software vulnerabilities, malwares, etc. However, there are important hardware vulnerabilities that

can be exploited to perform software or hardware attacks, leading to dangerous consequences.

Computers are increasingly handling sensitive data. The microprocessor is at the base of every type of computing node. In CPSs, the microprocessor is located in both the cyber elements and in cyber-physical elements. This component is thus crucial for any kind of system where it is embedded into. For this reason, the security concerns related to this component should be addressed carefully and extensively.

The focus of this section is on the microarchitectural structures within the microprocessor and the respective possible attacks. Also, the authors of [88], consider that microarchitectural channels are an important feature that requires investigation from the security point-of-view.

Nowadays, multiple processor cores are deployed in devices such as laptops and desktop PCs, servers and cloud-based system infrastructures as well as in mobile devices such as smartphones and wearables. All these devices, manipulate and store sensitive data. Side-channel attacks gained interest because the information can be leaked also through various hardware components of the microarchitecture. *Microarchitectural side-channel attacks* exploit information leakage that occurs within the microarchitecture a CPU. Usually, the structures within a microprocessor introduce important optimizations to improve the performances of the computing system. From the security perspective, however, the units within a microprocessor could be exploited from an attacker. The microarchitectural structures open up security criticality that if not addressed, might lead to security breaches. Thus, the whole microarchitecture might represents the attack surface. Evaluating side channel vulnerabilities of software running on a CPU should take the microarchitectural features of the processor itself into account [21]. Conversely to the earliest and to the majority of works concerning SCAs, microarchitectural side-channel attacks are mostly exploited through software [95], indeed are also called *software-based microarchitectural side-channel attacks*.

3.3.1 Attack points in Microarchitecture

According to [210], a microarchitectural SCA relies on the existence of a microarchitectural element with the following properties: (i) it is shared between attacker and victim application, (ii) its state changes based on the processed data and (iii) the state can be inferred from (a combination of) side channels. The first property usually defines the scope of the attack, because the element can be shared in many ways e.g., among some threads and some processes for L1 caches, among multiple CPU cores for Last Level Caches (LLC), etc. The second property imposes a dependence between the behavior of the element and the data that it is processing. Finally, the third property allows to "read" the internal state of the element through a measurable (directly on indirectly) side channel information.

The target of this section is not to exhaustively overview every possible attack in the microarchitecture, but rather to analyze the possible attacks directions and assess the potential risks they can present to the security domain. Several surveys available [4, 86, 88, 224] depict the current situation of microarchitectural SCAs. Indeed, this section provides a review of the state-of-the-art of the attacks targeting three microarchitectural features, i.e.,: (i) CPU Cache Memories in Sect. 3.3.1, (ii) Page-Translation in Sect. 3.3.1, and (iii) Speculative execution in Sect. 3.3.1.

CPU Cache Attacks

CPU cache is a fast and relatively small memory that buffers the data stored in the main memory, which requires higher latency time to access. The stored information is the one the CPU is most likely to need next. This expresses the so-called *principle of locality* and considers that usually the software does not access all its code or data uniformly. The term "locality" can refer to both time - *temporal locality* and space - *spatial locality*. This memory allows an increase of speed in the case the data needed is already stored in the cache (i.e., a *cache-hit* occurs) involves a penalty in case the data is not in the cache and requires in turn a load from another memory (i.e., a *cache-miss* occurs). Most recent CPUs include a hierarchy of caches, organized as in Fig. 3.4. The CPU cache is usually composed of different levels of caches of increasing size. Each cache is internally divided in blocks of fixed size, called *cache lines*. Further details about CPU architectures can

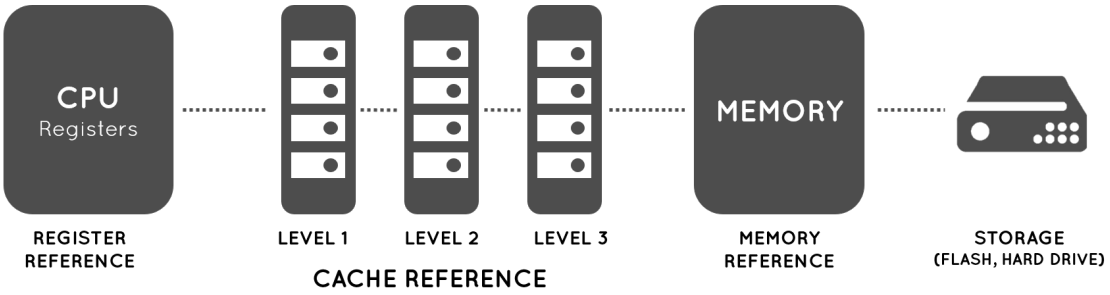

Figure 3.4: Memory Hierarchy representation.

be found on [105].

Cache side-channels exploit the fact that the memory is a shared resource at system level, i.e., it is not isolated by process, Virtual Machines (VMs), or privilege level. Many algorithms, including cryptographic algorithms, have memory access patterns that are dependent on the memory contents. Indeed, the ultimate targets of CPU cache attacks are usually (but not limited to) cryptographic information, such as cryptographic keys, that allow to violate the confidentiality of the system. There are essentially two basic mechanisms employed by an attacker to perform a cache side-channel attack against a victim:

1. the attacker sets the cache into a known state; it lets the victim work; it checks for changes from the known state.

2. the attacker establishes a timing baseline of the victim normal operation; it arranges some changes in the cache; it re-times the victim operation again.

In both cases, the attacks leverage the intrinsic property of caches, that is an access to data residing in the cache requires less time than the one to data not residing in the cache: when a time difference is detected, the attacker is able to tell if something has changed.

According to [4, 168], CPU cache attacks can be categorized in three types, based on the knowledge the attacker is able to collect about a target/victim process:

- Time-Driven [198]: the attack tracks the cumulative number of hits and misses in the cache, often by evaluating the overall execution time to get the aggregate profile of the cache activity (e.g., the total execution time for the victim to perform a block encryption).

- Access-Driven [168]: the attack leverages the ability to detect whether a cache line has been evicted, i.e., the information is related to the addresses accessed by the victim.

- Trace-Driven [5]: the attacker is able to capture the profile of the cache activity, i.e., he or she is able to tell the outcome of *each* memory access operation in terms of cache hits and misses.

In [88], general categorizations of the various attacks are presented and arranged according to different parameters, such as the sharing level of the resource exploited and on the level of concurrency used. Based on the employed attack techniques, the attacks can be divided in the following types: *EVICT+TIME*, *PRIME+PROBE* and *FLUSH+RELOAD*. These methods are discussed below.

EVICT+TIME

The technique *evict and time* considers the execution time measurement. Initially, an attacker lets the victim process to run in order to populate the memory and establish a baseline execution time. Then, the attacker proceeds to evict a cache set and lets the victim run again. But in this case, the attacker also measures the execution time of the victim process. If there is a variation between the two measurements, the attacker can infer that the cache set evicted has been accessed by the victim. The variation is due to a longer time required to fetch the needed data. Such methods require that the attacker is able to precisely detect the start and end time of a victim activity, which makes them usually not feasible when the attacker cannot control the victim process (e.g., to trigger a computation). This approach allows to establish which cache set is used by the victim.

This technique has been described and generalized in [178]. Bernstein, in the work [24], presents a variant of this technique, which still exploits the timing variability due to cache effects by relying on the existence of a statistical timing pattern of memory accesses during computations. Also this attack targeted the recovery of an AES encryption key. The victim is the OpenSSL AES implementation running on a remote server equipped with a Pentium III microprocessor. The same attack has been imporved by Bonneau et al. in [33]. The cache timing attack of Bernstein has been demonstrated on an embedded ARM-based platform also considering a virtualization-based system in [250]. Later, a ARM-platform has been targeted again with Evict+Time attacks in [217, 148]. A more recent work is reported in [122], which is performed using the Last-Level Cache (LLC). The LLC is shared between all cores and this is exploited by the authors to parallelize the attack leading to reduction of time needed to perform the attack.

PRIME+PROBE

The *prime and probe* technique consists of three steps: (i) in the first step (*prime* phase), the attacker fills a cache set with his own data; (ii) in the second step, the victim process is executed; (iii) in the last step, the attacker accesses again (*probe* phase) the data of the first step and measures the access time. By timing the access to its own data, the attacker is able to detect if the victim has evicted some portion of data. In this case, the attacker data has been replaced by the victim and accessing this data will incur in a cache miss. A cache miss will produce higher timing, while lower timing means that the cache set has not been replaced.

This technique, like *EVICT+TIME*, allows to determine which cache set is used by the victim, but improves the accuracy given that the cache access time is measured directly (conversely, in *EVICT+TIME* it is measured considering the whole execution time). This technique, requires that the attacker and the victim share the same cache sets.

Previous works [185, 178, 234] employ this technique by exploiting L1-Data (L1-D) cache to recover cryptographic keys. [185] targets the RSA implementation in OpenSSL, while [178, 234] attack the AES cipher of OpenSSL in Pentium E4 and Athlon64. The works [3, 265] exploit instead the L1-Instruction (L1-I) cache. Implementations of ciphers with key-dependent instruction flows can be vulnerable to the attacks exploiting the L1-I leakage. Knowing the changes of state in the instruction cache might allow an attacker to extract the instruction flow of the victim software. The authors of [3] are able to mount and attack OpenSSL-DSA implementation to successfully recover DSA keys. The attack is performed in real-world settings, i.e., without modifying the cipher process. Other studies [150, 130, 199] employ the *PRIME+PROBE* method on the LLC. In [150] the target is the implementation of ElGamal in GnuPG. [199] is based on LLC, but in a virtual machine environment. The attacker is able to obtain confidential information, such as network traffic load estimation and keystrokes timing attack in a physical

infrastructure shared among different users also in the case their actions are isolated through machine virtualization.

FLUSH+RELOAD

The *flush and reload* technique is a variant of *PRIME+PROBE*. The attack consists of three steps, where a malicious process (or spy process) monitors the memory lines accessed by a victim process: (i) in the first step, the monitored memory line is intentionally flushed from the cache; (ii) in the second step, the spy waits for the victim to access the memory line; (iii) in the last step, the spy reloads the memory line, measuring the time required to load it. A fast reload means that the victim accessed this line (and reloaded it) during step (ii). A slow reload states the opposite, because the line will need to be fetched from memory and the reload will take much longer time.

The *FLUSH+RELOAD* technique relies on sharing pages between the spy and the victim processes. The advantage with this technique is that the attacker can achieve finer granularity, with respect to *PRIME+PROBE* and *EVICT+TIME*, because it works on single cache line.

This technique was first employed in the work of Gullasch et al. [98] attacking the AES cipher. However, the first work assigning the name *FLUSH+RELOAD* was by Yarom et al. [258]. This study is directed towards the LLC and uses the *clflush* instruction to evict the monitored memory locations from the cache. This work extends the one of Gullasch et al. [98], by allowing cross-core attack (i.e., the spy and the victim processes are executed in parallel on different execution cores) and virtualized environments. The attack targets the RSA implementation of GnuPG. This technique was also employed to recover AES encryption key in [119, 120, 99, 96] on Intel x86 platform and on ARM in [148]. Apart from being employed against cryptographic algorithms, this technique has been employed interestingly in the detection of user actions in [148, 247]. The authors of [247] use the SCA on graphic libraries and their attack able to detect the keystrokes and discover what keys have been pressed, leaking sensive information (e.g., PIN or password) of an user. The attack is carried out on a PC as well as on a mobile device.

It is worth mentioning that there exist variants to the *FLUSH+RELOAD* technique, i.e., *EVICT+RELOAD* and *FLUSH+FLUSH*. In *EVICT+RELOAD* [96] the steps are the same of *FLUSH+RELOAD* technique, but it avoids the use of *clflush* instruction which might not be always available. Instead, the eviction operation is performed. However, this technique leads to lower accuracy and it requires longer time to be completed. *FLUSH+FLUSH* [97] is another variant: it still exploits timing variations of the *clflush* instruction, but instead of measuring the time to reload the data, it measures the time needed to flush the data. It has been employed in [37] to recover the encryption key of OpenSSL AES implementation.

Page-Translation Cache Attacks

The memory addresses used by a processes do not correspond directly to actual storage locations, but are virtual addresses. Virtual addresses are mapped by the operating system into physical addresses. Address translation is the mechanism for mapping virtual to physical addresses. Address translation provides a primitive security mechanism, i.e., isolation: each process has a virtualized view of memory with limited visibility/access to the underlying memory space. The mapping between physical to virtual addresses is stored in data structures called pages. Every memory access realized by a process is translated[1], affecting negatively the performances, because of the penalty introduced with two memory accesses: the first access to obtain the physical address and a second one to get the actual data. Using the principle of locality Exploiting the *principle of locality*, to store recently translated page addresses improves the performances. This special address translation cache is referred to as a Translation-Lookaside Buffer (TLB). Before translating a referenced address, the processor checks the TLB to identify the physical page corresponding to the virtual page, or to detect that the needed page is not cached [105].

Basically, the TLB is a special type of cache itself does not store data nor instructions, but instead stores virtual addresses, physical page frame numbers and other metadata (such as dirty bit, valid bit, etc.). Indeed, like the Instruction or Data CPU Caches, it exhibits different timing behaviors between cached and uncached addresses. By measuring the time required to access an address, an attacker is able to determine whether the respective translation is present in the TLB [210].

The authors of [94] present an attack called *TLBleed*[2] able to leak a cryptographic key of EdDSA and RSA in *libgcrypt*[3]. In this attack, both the malicious and victim process are scheduled on the same core and share the same TLB. The first attack exploiting TLB [113] defeated the Address Space Layout Randomization (ASLR). The local attacker with restricted privileges exploited the memory management system to deduce information about the privileged address space layout. To prove the efficacy, the attack has been performed on both 32 and 64 bit x86-based CPU architectures. Also the work [93] targets the ASLR, but employs *EVICT+TIME* technique and enforces the attack de-randomizing virtual addresses from the web browser process. In [208], *EVICT+TIME* technique is employed against the Memory Management Unit (MMU) to reverse engineer the size and internal architecture of page table caches on 20 different microarchitectures.

---

[1]The system needs to support virtual memory as memory management technique

[2]OpenBSD disabled Hyper-Threading completely, disabling this vulnerability at a large cost to processor performance. See: https://www.mail-archive.com/source-changes@openbsd.org/msg99141.html

[3]https://gnupg.org/software/libgcrypt/index.html

Speculative Execution Attacks

Modern microprocessors employ several methods to achieve high performances. Among them, speculative execution is one of the main techniques employed. The underlying principle of speculative execution is that the instructions are executed before they are required. Instead of executing the instructions in order, they are executed as soon as the data is available. In case an operation (e.g., cache miss) stalls the execution of an instruction (impacting on performances), to minimize the amount of time it spends waiting for the data, the CPU can execute instructions after the stalled one, essentially reordering the code in the program (i.e., *out-of-order execution*). In this way, the performance can be increased. In absence of speculative execution, the processor would have to wait previous instructions to be completed before executing successive ones (i.e., *in-order execution*). However, in a conditional branch the processor might not be able to tell which branch will be taken, so it will try to predict which branch to take. The prediction outcome is performed by the Branch Prediction Unit (BPU) considering the most likely path, based on the previous history stored in the Branch History Buffer (BHB). When the branch instruction is evaluated, it could be possible that the speculatively executed instructions would not be needed and in this case their results would be discarded.

In the last few years, a new series of side-channel attacks, called *Speculative Execution Attacks* or *Transient-execution attacks*, have been discovered that might allow access to sensitive information. All these microarchitectural attacks exploit speculative execution features of modern CPUs to leak information. In case the processor is speculatively executing instructions, those to be discarded still generate state changes in the microarchitecture.

Initially there were three variants of the issue:

1. Variant 1 - Bounds Check Bypass (CVE-2017-5753)
2. Variant 2 - Branch Target Injection (CVE-2017-5715)
3. Variant 3 - Rogue Data Cache Load (CVE-2017-5754)

These security issues are commonly known as Meltdown [149] (Variant 3) and Spectre [137] (Variant 1 and 2). However, they do not represent single attacks, but rather a class of attacks. Indeed, after their initial disclosure[4], several variants appeared [106, 42].

Meltdown

Meltdown was first presented in [149]. It exploits speculation caused by out-of-order

[4] https://googleprojectzero.blogspot.nl/2018/01/reading-privileged-memory-with-side.html

execution to attempt to read kernel memory from user space without misdirecting the control flow of kernel code. In this attack, an illegal instruction accessing to kernel memory causes a fault. If this instruction is speculatively executed, it leads to an unauthorized access and triggers a fault. Although the violation still occurs, that instruction returns data and following instructions might not be committed but nevertheless might change the microarchitecural state. Finally, the attacker can perform a side channel (e.g., *FLUSH+RELOAD*) to obtain the data leaked.

Differently than Spectre, Meltdown does not use branch prediction, but it relies on out-of-order execution of the instructions successive to the instruction causing a fault. Moreover, it exploits vulnerability specific to Intel and ARM processors that allow to bypass memory protection during speculation [149].

Spectre

Like Meltdown, Spectre represents a class of transient-execution attacks and it is built on similar techniques. Also Spectre involves inducing a victim to speculatively perform operations which leak the victim information through a side channel. It exploits branch prediction features.

The first variant of Spectre exploits conditional branches: initially an attacker mistrains the CPU branch prediction into taking the same correct branch. After that, it triggers a fault accessing the out-of-bound area. However, the prediction is distorted and because of speculation the transient instruction is executed, provoking the leak. The instruction is reverted but the leak remains.

Spectre variant 2 exploits indirect branches: The basic idea for the attack is to target victim code that contains an indirect branch whose target address is loaded from memory and flush the cache line containing the target address out to main memory. This will force the CPU to reload the cache line. While reloading, the CPU will speculatively execute instructions based on branch prediction. If the attacker mis-trains the branch predictor with malicious destinations, the speculative execution continues at a location chosen by the opponent. Also in this case, the leak remains.

Both variants of Spectre were initially presented in [137]. Spectre attacks exploit transient execution following control or data flow misprediction, while Meltdown, exploits transient execution following a faulting instruction. With respect to Meltdown, Spectre attacks work on a wider range of processors including most AMD and ARM processors [137].

These attacks are critical, especially for important systems employed in critical infrastructures, because the vulnerability is brought by the speculative execution that is found in almost any microprocessors of Intel, AMD and ARM vendors. This represents a serious threat to any safety and security critical systems such as Critical Infrastructures, considering those microprocessors are used in billions of devices. Moreover, recently, other variants have been discovered [42], which makes difficult to apply security patches.

In general, hardware threats are of considerable concerns because they can target many different devices and that the underlying security hardware vulnerabilities can take several months or even years to be fixed even though a patch has been discovered.

3.4 Attacks on FPGA

Considering the reconfigurable technologies introduced in Sect. 2.5 and the Information Security properties of Sect. 3.1, this section analyzes the state-of-the-art on security issues related to the bitstream employed to describe an IP (Intellectual Property) core, which contains the configuration data employed to program an FPGA.

The remainder of this section is an extended and revised part of the publications [47, 48].

3.4.1 Introduction

A general overview of the feasible attacks against FPGAs, as well as a description of security issues and open problems regarding system security of FPGAs is provided by Wollinger et al. in [254]. Security features, such as anti-tampering and data protection techniques, are described in [233]. In [76], the authors define an FPGA threat model and evaluate how the security features offered by most FPGA vendors address the threats.

3.4.2 Bitstream Confidentiality

Guaranteeing bitstream confidentiality implies to protect from eavesdropping performed by external sources such as attackers or competitors. In this way, the information exchanged has to be maintained secure in the sense that it is not disclosed to unauthorized third parties. To maintain the confidentiality it is sufficient to encrypt the data.

Nowadays, bitstream confidentiality can be achieved resorting to the integrated bitstream encryption mechanisms offered by most FPGA manufacturers [12, 142, 251, 118]. [34, 145, 6, 12, 142, 256, 251, 118]. These techniques allow to assist system designers in the protection of the secrecy of their IP cores, preventing the product to be cloned or reverse engineered.

The bitstream encryption operation is based on symmetric cryptography, i.e., the key used to encrypt the information is the same used during the decryption. The encryption key is generated by the system designer at design time and stored inside the FPGA. The device decrypts the incoming bitstream during the configuration phase. The FPGA stores the encryption key internally in a devoted memory, which

could be backed-up by a battery (e.g., BBRAM - Battery backed RAM) in order to maintain its content or a non-volatile one-time-programmable memory. The memory storing the key is designed to prevent physical attacks. However, the encrypted bitstream configures the entire device. To decrypt partial bitstreams, system designers should build their own decryption engine requiring additional logic, thus reducing the usable area. [7] presents three methods for secure partial bitstream relocation.

Bitstream encryption is an effective solution to protect designer's IP against cloning or reverse engineering and IP disclosure [171]. However, in some cases encryption may not provide sufficient security, as reported in [163, 164]. Another approach to protect IP cores against piracy and reverse engineering can be obtained through obfuscation. The authors of [128] leverage FPGA dark silicon to obfuscate the functionality of the design. Other works discuss techniques to consider for bitstream confidentiality as well as authentication[75, 230].

3.4.3 Bitstream Integrity and Authenticity

Since encryption alone does not protect the bitstream from modifications, integrity aims to ensure that the data exchanged does not undergo to modifications carried by unauthorized third parties across the network links during the transfer. Moreover, it provides assurance also from unintended modifications that might corrupt the data due to errors, e.g., transmission errors. If the integrity is maintained, also the system receiving the bitstream preserves its integrity.

Error control against data corruption can be accomplished with error-detection techniques, such as checksums and Cyclic Redundancy Checks (CRC). To confirm that the configuration data stored in an FPGA device is correct, most vendors offer dedicated hardware for CRC features [13, 257]. However, CRCs are not suitable to protect against intentional malicious alteration of data [219] since employ reversible functions. Moreover, they are vulnerable to collision attacks, thus they do not guarantee adequate security levels.

Hashing is another method commonly used to validate the integrity of information using a one-way function, i.e., infeasible to invert. The fixed-length output of a cryptographic hash function, called message digest or simply hash, can be thought of as the fingerprint of the input bitstream, offering strong collision resistance. Cryptographic hashing primitives have been employed in [179, 180, 75]. Message digest, however, does not authenticate the source. To verify the origin of the bitstream, the secret key shared between the vendor and the device can be combined together with a cryptographic hash function to generate a Hash-based Message Authentication Code (HMAC). In [180] different authenticated encryption schemes have been evaluated and the dual-pass Counter with CBC-MAC (CCM) has been identified as the best option by lowest area footprint. However, Dual-pass authenticated encryption algorithms separate authentication and encryption

procedures and therefore require significant overhead. Usually, authentication and confidentiality aspects are considered together, as proposed in [75, 19].

In presence of DPR, specific solutions to security problems, as well as safety [65], have also been discussed in literature. In [34] a flexible security system based on bitstream encryption is proposed. While using FPGAs DPR, designers can freely choose encryption/decryption algorithms implemented as reconfigurable modules. In [261] authors propose two schemes based on a hard-wired PowerPC processor core and the MicroBlaze soft-core processor to perform a secure DPR. For authentication and encryption phases SHA-1 and AES algorithms are used respectively, implemented as C programs. Time comparative table shows that in both schemes, the total processing time is not sufficient for practical partial reconfiguration. In [109] authors developed a secure DPR system based on encryption of partial bitstreams with AES-GCM cipher. AES-GCM is an authenticated encryption cipher which guarantees both the confidentiality and the authenticity of a message. In [129] is shown an improvement of the security of DPR in FPGAs re-encrypting a remotely received bitstream with a unique random key, while providing low area overhead and a high reconfiguration throughput.

3.4.4 IP Licensing and Activation

Providing confidentiality, integrity and authenticity of a bitstream protects the end-user device from malicious attackers. However, the intellectual property is left unshielded from piracy, which could damage the related business model of an application. IP theft introduces problems with design rights associated to the soft IP cores. Encryption and obfuscation are strong tools to ensure security, but they cannot protect the IP in every stage of the life cycle. Several works address these challenges [152, 123, 264, 104, 152, 123, 264]. Enforcing IP security can be achieved resorting to solutions from a related area, i.e., trusted computing. In particular, the Trusted Platform Module (TPM) is a microcontroller used to authenticate a target platform, enabling several cryptographic features [236]. The TPM is embedded in and interacts with the target platform, providing cryptographic primitives for secure key generation and storage, random number generation and remote attestation. In literature, different methods of activation and licensing for IP cores protection have been employed: in [100] a new scheme to track and control the licensed designs is presented, adopting public-key cryptography and symmetric encryption as well. The authors of [262] propose an IP protection mechanism to restrict IP execution only on specific FPGA devices, limiting unauthorized copies and integration of the IPs. This solution, together with [123], enforces a pay-per-device licensing scheme for system developers, instead of purchasing unlimited IP licenses. Finally, a recent work [15] provides a remote licensing and activation mechanism, guaranteeing anonymity for the end users.

Chapter 4

Performance Monitor Counter Attacks

Microarchitectural side-channel attacks represent a dangerous threat for Cyber-Phisical Systems. Security countermeasures need to be carefully evaluated, because their integration might interfere with other design aspects of the system. This chapter describes the implementation of a such type of attack in a safety and security critical CPS architecture and details a mechanism to effectively neutralize the attack, without affecting the safety domain. This chapter revises and extends the past publications [45, 46].

4.1 Introduction

Cyber-Physical Systems (CPSs) are the root of a fourth industrial revolution [51]. *"Cyber-physical systems are physical and engineered systems whose operations are monitored, coordinated, controlled and integrated by a computing and communication core"* [196]. A CPS integrates processing units, sensors and actuators, enabling the interaction of the computing infrastructure with the physical world. The Internet of Things (IoT) forms a foundation for this cyber-physical systems revolution [51]. All devices of a CPS and different CPSs are interconnected in order to create a network enabling billions of systems and devices to interact and share information. Thanks to their ability of transforming traditional processes by integrating technologies from various sectors, CPSs are bringing innovation to many industries including: automotive and aerospace, chemical processes, smart energy and water grids, healthcare, manufacturing and transportation [26, 55, 196, 248]. Several of these application domains involve the control of critical infrastructures providing services that constitute the technological backbone of our society [69]. These infrastructures can damage themselves, people, or properties when they are improperly used [253], and this improper use can be either the result of a failure of

one of their components or an intentional attempt to corrupt their behavior.

Safety and security are therefore two critical properties of every CPS, both sharing identical goals: protecting the CPS from hazards due to accidental failures (safety) or due to intentional attacks (security)[203, 172]. In this context, there is a recognized request to consider them under a unified view when designing and operating complex CPSs [189, 187, 203, 253]. This is particularly important every time a security mechanism may negatively impact the safety of the system or vice versa [188].

This chapter analyzes in a specific scenario how the Performance Monitor Counters (PMCs) available in several commercial microprocessors may have severe implications on the interplay of safety and security of a CPS.

PMCs can be used for several purposes including performance modeling and optimization, debugging, benchmarking, and in-field monitoring (see Section 4.2). The integration of computing and physical elements in a CPS introduces a vast range of design and operational constraints. Among them, CPSs often require to operate under real-time constraints [253]. PMCs are an effective instrument to detect timing violations in multiprocessor systems. The timing of different tasks can be profiled by recording time related PMCs (e.g., the Clock Cycle Counter - CCC and the L1 Data Cache-Miss counter - DCM in the Intel architectures) over several executions in order to build a model of the behavior of the system. This can be done either by setting simple thresholds [79, 175, 159] or by exploiting machine learning models [1]. The model can then be used at run-time to detect anomalies.

However, time related PMCs have been exploited to perform different classes of attacks (see Section 4.2). These PMCs can be used to implement the side-channel attack described by Bonneau and Mironov in [33], which is able to discover the secret key of the Advanced Encryption Standard (AES) cipher.

Sect. 4.4 proposes a mitigation strategy able to increase the complexity of this attack and discusses its interplay with the effect on a selected safety mechanism. Among the different safety techniques against timing violations, the methodology proposed by Esposito et al. in [79] is here considered for its semplicity. This technique builds a simple timing model for different tasks based on two thresholds.

The proposed attack mitigation technique is based on the application of a PMC poisoning schema. The poisoning alters the value distribution of the PMCs in such a way to harden the work of the attacker while preserving those properties that allow detecting timing violations in the system.

Based on the concepts presented here, different poisoning schemas can be derived in order to work with different monitoring techniques. However, it is hard to generalize the impact of the poisoning on each schema. This must be evaluated case by case.

A set of experiments was carried out to evaluate the impact of the proposed attack mitigation technique on the safety and security of a sample node executing different tasks. Experiments consider both synthetic tasks hard to monitor and

tasks taken from the MiBench benchmarks [101]. The node uses the AES cipher (victim of the attack) to encrypt information. MiBench benchmarks were selected since they implement a set of typical algorithms employed in several embedded systems and have been also used in several reliability evaluation studies [242, 240, 244, 243, 54, 207, 124]. Results show the capability of the proposed technique in increasing the complexity of the attack considering both the CCC and the DCM counters, while introducing a low impact on the timing violation detection capability of the system. The technique proposed here outperforms results published in [46] both in terms of better security and reduced impact on the safety of the system.

4.2 Performance Counters

PMCs are special registers available in most microprocessor architectures. They allow monitoring of several classes of events including branch predictions, cache hits/misses and process timing. Monitoring of events through PMCs has a variety of uses in application development, including performance modeling and optimization, debugging, and benchmarking [173, 162, 73, 125]. The privileges required to access these counters depend on both the processor architecture and the operating system (OS). However, during application development, the designer has usually full control on the development environment and a full access to the available PMCs does not represent a security threat.

When moving to in-field applications, access to PMCs requires to carefully consider the interplay between safety and security.

A set of applications evaluate predefined PMCs signatures in order to detect failures or attacks. PMCs can be exploited as a valuable tool for the development of software based self test (SBST) routines [231]. The capability of monitoring cache-misses through PMCs (DCM counter) was used in [181] to perform SBST of cache memories. In this type of approaches, an exact value of the counter is expected at the end of the test routine. PMCs signatures were successfully exploited to detect firmware modifications in a CPS controlling a power grid in [249]. For this class of PMC uses, poisoning techniques such as the ones presented in Sect. 4.4 cannot be applied, since they would prevent the deterministic behavior of the counters for selected processes.

Nevertheless, there is a large class of techniques that try to build models of the behavior of the system based on statistical off-line PMC profiling. These models can be used in-field to detect different types of anomalies.

Xia et al. employed the PMCs to monitor the control-flow integrity of software applications [255]. By analyzing the branch instructions behavior, they were able to detect deviations from the correct control flow.

Yilmaz [259] proposed a technique for faults localization in software applications. By off-line profiling the number of executed instructions considering correct and faulty executions, they were able to build a behavioral model of the software able to detect in-field applications differentiating from the model. A similar technique from the same authors based on the monitoring of the function execution time was also presented in [260].

The authors of [175] used the PMCs to estimate the Worst-Case Execution Time (WCET) for safety-critical applications.

In [159], WCET-aware Performance Monitoring Units were proposed for safety certification in the automotive domain.

In [79] PMCs were used to detect faults causing deadline violations in multi-core systems.

A set of 10 PMCs was analyzed in [1] to build a machine learning model based on Supported Vector Machines (SVM) able to detect different types of anomalies.

In this category of approaches, poisoning techniques such as the one presented in Sect. 4.4 can potentially be applied. Nevertheless, the actual impact of the poisoning strongly depends on the target technique and it is hard to generalize. For this reason the focus is on a selected technique presented in [79] to show the interplay between safety and security in a specific case.

When considering the use of PMCs to perform attacks, they can be also used as a side-channel leakage source to steal sensitive information. Most related publications focus on the cache behavior during encryption with the AES algorithm. Bernstein was able to remotely recover the complete AES key exploiting timing information related to cache accesses [24]. Bounneau and Mironov performed a similar attack with a reduced number of samples to recover the AES key when applied to Intel architectures [33]. PMCs were employed as source of side-channel information also to attack encryption algorithms on AMD platforms in [237]. Side-channel attacks were possible also for asymmetric key cryptography, as reported in [25]. The attack, carried out on Intel platforms, targeted a 1024 bit key of RSA and exploited the monitoring of branch-miss events.

Proper defense measures can be taken if the attack is detected. The authors of [9] proposed a generic detection mechanism, using a pre-trained classifier, able to deal with a variety of micro architectural side-channel attacks, including also cache-based attacks. However, rather than reacting to an attack, it is important to work on proactive techniques able to prevent or increase the complexity of the attack.

The works proposed in [45] and [46] that are the base for the research presented in this chapter represented the first attempt to protect both the CCC and DCM counters against the Bonneau and Mironov attack taking into account the impact on the safety of the system.

4.3 CPS Architecture

4.3.1 CPS and node architecture

Fig. 4.1 shows the general CPS architecture considered in this chapter, which is a typical *Supervisory Control and Data acquisition* (SCADA) system. *Computing nodes* perform local computations and are responsible for controlling a network of sensors and actuators managing the operations of the physical infrastructure. Special *monitoring nodes* coordinate the work of a set of computing nodes by constantly exchanging data and information with them. Depending on the application, CPSs provide a wide set of specifications against which the systems must operate. Several manufacturing plants employ nodes based on low-end microprocessors either running bare-metal applications or old desktop operating systems [253]. However, the increasing complexity of the controlled infrastructures is quickly moving these systems toward more complex microprocessor architectures [103].

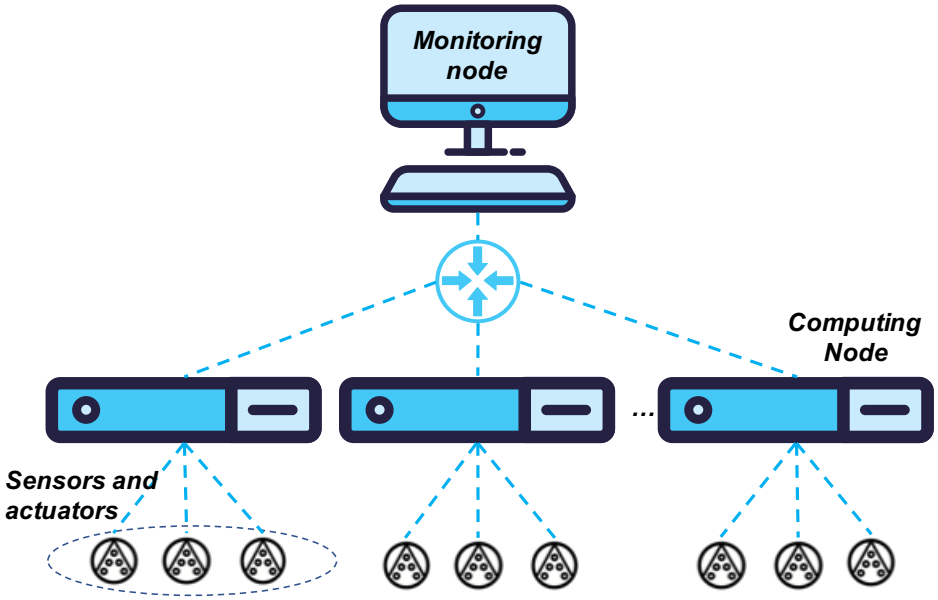

Figure 4.1: Generic architecture of the considered CPS. Figure taken from [45]. ©2019 IEEE.

Fig. 4.2 shows the conceptual architecture of a generic computing node.

The node runs an operating system providing a set of services required to accomplish different application tasks. The number and type of tasks depends on the available sensors/actuators and on the function the node has to implement. Moreover, tasks must be executed under given timing constraints [253]. Overall, the goal of the executed tasks is to acquire data from sensors, elaborate raw data and encrypt/decrypt payloads to communicate with other nodes. Data exchanged with external nodes (e.g., monitoring nodes) are encrypted with a symmetric key by an

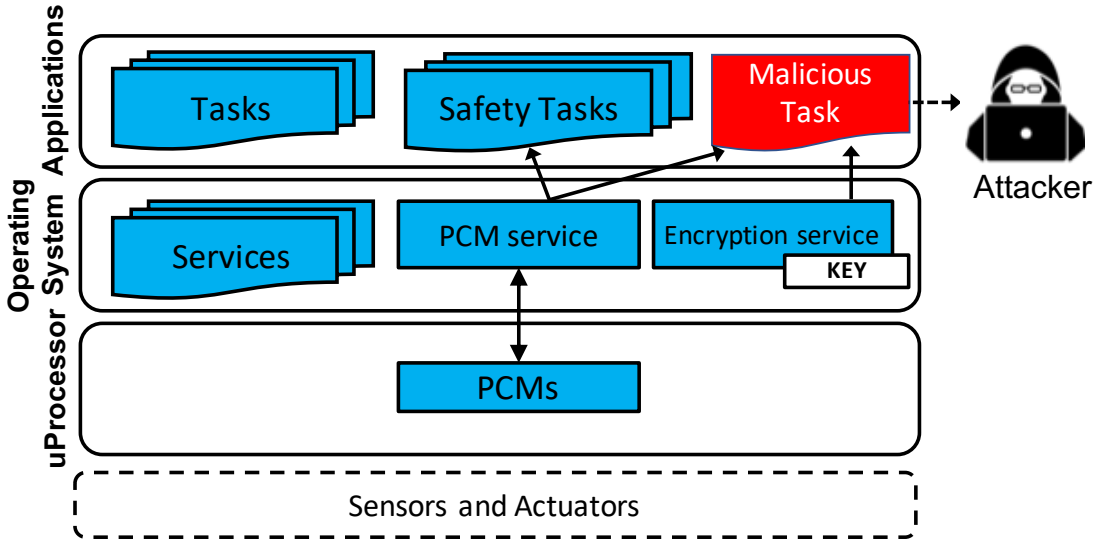

Figure 4.2: Generic architecture of the considered computing node. Figure taken from [45]. ©2019 IEEE.

appropriate service module integrated in the OS. In general, any task running at the application level can request the OS services.

4.3.2 Safety task

As discussed in Section 4.2, different safety mechanisms to control the correct operation of the node can be implemented as additional safety tasks at the application level (Fig. 4.2). The safety task considered here (based on the technique proposed in [79]) exploits PMCs profiling and on-line monitoring to detect faults causing abnormal execution times of a task. Overall, the considered safety technique consists of two phases named: (i) *off-line phase* and (ii) *on-line phase*.

During the off-line phase, each task is profiled over several executions in order to analyze the distribution of the values assumed by the considered PMCs. Two PMCs strictly related to the execution time of an application are considered in this study: the Clock Cycle Counter (CCC) and the L1 Data Cache-Miss counter (DCM).

In principle, the value of a PMC for a given task with given inputs and execution environment, should provide deterministic and reproducible values. However, in complex multi-core systems as the ones considered here, several complex often-unknown HW/SW interactions that are hard to predict (e.g., out-of-order execution models in which instructions are executed in a non-deterministic order, the memory hierarchy featuring different levels of cache memories that determine non-deterministic data access profiles, bus arbitration and in general all controllers,

etc.) may introduce variability in the PMC values. Different inputs across different executions of a task are another source of variability of the observed PMCs. Therefore, the values measured for a PMC across different executions can be considered as a random variable X, characterized by an empirical Cumulative Density Function (CDF), $F_X(x)$ (see Fig. 4.3).

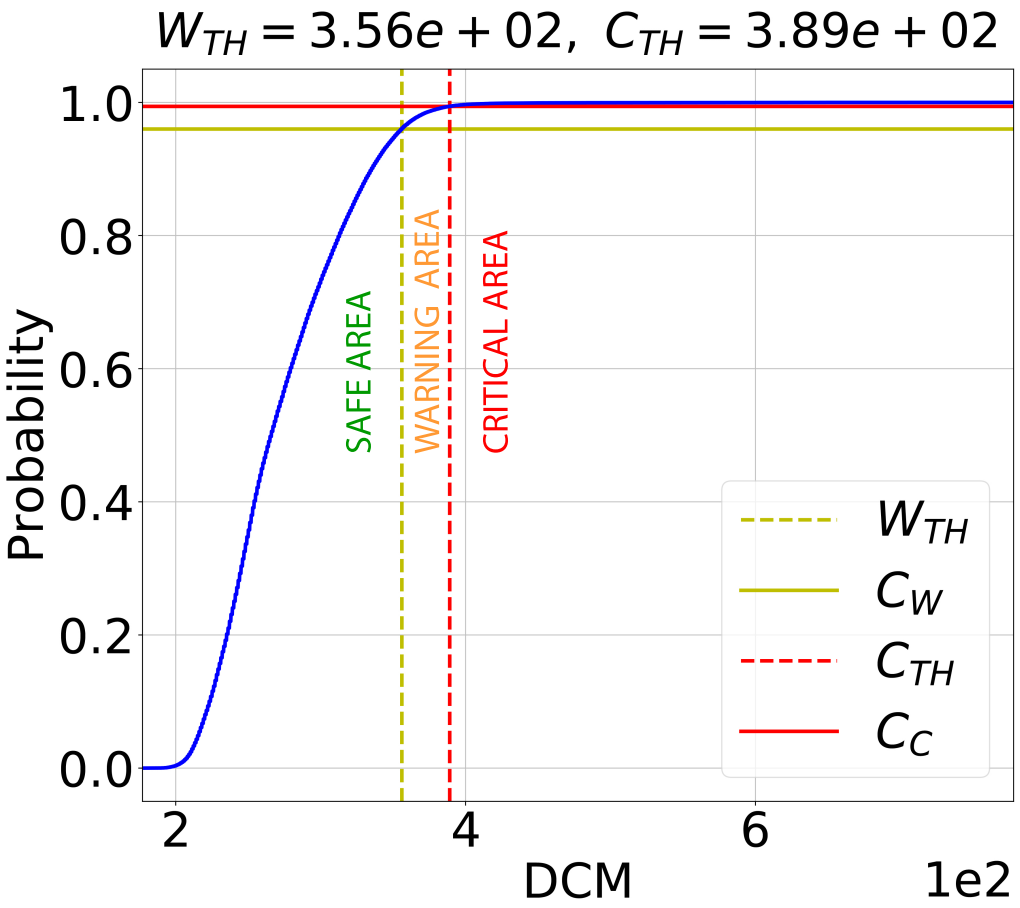

Figure 4.3: CDF of DCM counter of the Susan-Smoothing MiBench benchmarks [101] computed profiling 100,000 executions of the task. Figure taken from [45]. ©2019 IEEE.

Based on the collected profiles, each task can be associated to three operating areas reflecting the state of the system: *safe area*, *critical area* and *warning area*. Two thresholds are defined to separate the aforementioned operating areas: W_{TH} and C_{TH} (see Fig. 4.3). These thresholds are chosen by selecting the confidence level for the warning area (C_W) and for the critical area (C_C). Confidence levels can be chosen arbitrarily during the design phase. Strict confidence levels increase the number of false positives and the performance overhead due to a larger number

of recovery operations. Wide levels may not detect all failures of the system. In details, W_{TH} and C_{TH} can be computed by solving the following inequalities looking at the collected profiles:

$$P(X > W_{TH}) < C_W \Rightarrow F_X(W_{TH}) > 1 - C_W \tag{4.1}$$

$$P(X > C_{TH}) < C_C \Rightarrow F_X(W_{TH}) > 1 - C_C \tag{4.2}$$

During the on-line phase, the safety task monitors the PMCs of every task in order to determine in which area the execution can be mapped. The task state is then classified as follows:

- *safe direct*: if the value of the PMC is below the warning threshold.
- *critical direct*: if the value of the PMCs is above the critical threshold. In this case a recovery action must be issued;
- *warning*: if the value of the PMCs is between the warning and the critical threshold. In this case, if this conditions is detected α consecutive times the task is marked as *critical warning*, otherwise it is classified as *safe warning*.

The choice of α is related to the probability that the system is in a *safe* state after α consecutive *warning* classifications denoted as $P(FP_\alpha)$. $P(FP_\alpha)$ is obtained during the process profiling of the off-line phase. According to [79] α can be calculated as:

$$\alpha = \frac{ln(1 - P(FP_\alpha))}{ln(F(C_{TH}) - F(W_{TH}))} \tag{4.3}$$

4.3.3 Attack model

Securing the CPS architecture presented in Fig. 4.1 is an important task. The attack model considered focuses on an attacker interested in recovering the AES encryption key of a node to carry out malicious actions. We suppose the attacker is not interested in denial-of-service attacks, because they disrupt the offered services and prevent the control of the CPS. A successful attack on a node may spread the infection to every node, thus compromising the whole system. In the case all nodes share the same secret key, the whole system would be immediately compromised when a single node is compromised. When the secret key is different for every node, the same malicious task could target another node and the attack could be repeated until all nodes composing the CPS are under control of the attacker.

Looking at the architecture of the node reported in Fig. 4.2, we assume that enough effort has been carried out to secure the hardware OS level of the node. This includes securing the secret key and the encryption/decryption service. Nevertheless, we assume that the attacker may exploit user level vulnerabilities to inject a malicious task (e.g., a virus or a malware) within a node. The attack could be

undertaken on a specific node because the attacker could have gained physical access to it, or because that specific node offers unique vulnerabilities. The malicious task is a user application that can exploit the computational resources of the node as well as the services offered by the OS of the node. Therefore, the attacker can probe the PMCs, trigger the encryption process and send the information to a remote entity that can process them off-line in order to perform the attack.

Despite PMCs are a force point from the safety perspective, they represent a weak point from the security standpoint. They expose the system to *timing attacks*, a category of side-channel attacks [24]. PMCs can therefore be considered as a double edged weapon. The attack here reproduced is the side-channel attack presented by Bonneau and Mironov in [33], which targets the AES encryption algorithm.

The theory exploited to perform the attack is based on the concept of *cache-collisions* during the final round of the AES encryption cypher.

For performance reasons, the algebraic operations of a software-implemented AES cipher are combined in precomputed values stored in different lookup tables. Thus, the encryption can be considered as a sequence of table lookups. As all data of a program, these tables are loaded in the L1 data cache memory during the encryption process. If the data is already loaded in the cache memory, a lookup produces a cache-hit. On the other hand, when the data cannot be found in the cache memory, the lookup generates a cache-miss, which will take on average more time to be served since it requires to access data from a slower memory level. Depending on the organization of the cache, each cache line may store multiple table entries. A *cache-collision* occurs when a pair of different lookups targets the same cache line and for sure does not generate a cache-miss.

Being able to detect when a collision occurs is important for the attacker, since it gives a clue of which element of the table has been accessed.

Let us denote with i and j the index of two bytes of a generic 16 bytes ciphertext ($0 \leq i, j \leq 15$), with c_i and c_j their respective values, and let us consider the encryption time (i.e., CCC counter) as available information. As described in [33], the first goal of the attacker is to record in a data structure the timing data for random ciphertexts (samples) for several pairs (c_i , c_j). The data structure is organized as follows:

$$t[i, j, \Delta] = CCC_k \tag{4.4}$$

where $\Delta = c_i \oplus c_j$ and CCC_k is the encryption time of the k^{th} collected sample. If multiple data are collected for the same entry of the table, the average time is recorded. According to [33], if a cache-collision occurs the following equality becomes true:

$$k_i \oplus k_j = c_i \oplus c_j \tag{4.5}$$

As discussed before, when a collision occurs, the encryption time should be significantly lower that the cases in which there is no collision. To succeed in the attack, the attacker has to find one value $\Delta'_{i,j}$ for each i, j such that:

$$t[i, j, \Delta'_{i,j}] < \overline{CCC} \tag{4.6}$$

where $\overline{CCC}$ is the average encryption time over all collected samples. We denote this as *collision condition*. In this case $\Delta'_{i,j}$ becomes an accurate guess for the true value $k_i \oplus k_j$. This reduces the space of possible values of $k_i \oplus k_j$ that an attacker has to test to recover the key. Several further optimizations are proposed by [33] to reduce the number of samples required to perform the attack. Nevertheless, the general concept of the attack remains the same.

When using the CCC timer, the attacker exploits the timing as an indirect measure of the number of cache-misses. The correlation between the CCC timer and the number of cache misses is the key factor. In the Pentium 3 processor analyzed by [33] this correlation was very high and therefore a low number of samples (2^{16}) was enough to perform the attack. With a more complex Pentium IV Xeon processor, [33] found that, due to a lower correlation, the number of samples required to perform the attack increased to 2^{19}. We analyzed this correlation for the Core i7 processor considered here. Due to the complexity of this processor, there is very low correlation between the timing and the number of cache-misses. This means that most of the collected samples are actually not useful for the attack and a significantly higher number of samples is required to recover the key (2^{27}). Nevertheless, the attack is still possible.

Based on this consideration, if a direct measure of the number of cache-misses is available to the attacker (through the DCM counter), a significantly lower number of samples should be enough to recover the key as confirmed by the experimental results provided in Section 4.5.

The possibility to access the PMCs by the malicious task, that includes timing and cache access information, opens a path to properly implement this attack in the node architecture presented in Fig. 4.2.

4.4 Attack mitigation

The success of the attack introduced in Section 4.3.3 depends on the PMC samples collected by the malicious task. To achieve its goal, the attacker has to statistically analyze the distribution of the different samples.

The main idea here proposed to counteract this attacker is to modify the PMC service implemented at the OS level of Fig. 4.2. Each PMC reading is poisoned in order to obfuscate the statistical properties of the PMC:

$$PMC' = cf(PMC) \tag{4.7}$$

where PMC' is the corrupted PMC reading, PMC is the correct PMC reading, and $cf(PMC)$ is the *corruption function* that is a function of the value of the counter.

However, this corruption may jeopardize the capability of the safety task described in Section 4.3.2 to detect timing violations. This may potentially create both false negatives (i.e., undetected time violations) or false positives (i.e., safe conditions detected as violations). Therefore the corruption level must be carefully considered both from the security and from the safety standpoint.

It is worth recalling that, in the proposed architecture, the PMC service is considered secure (see section 4.3.3) and represents the only user access point to the PMCs.

There exist several possibile corruption functions. In Sect. 4.4.1 and Sect. 4.4.2 are considered two different corruption functions, deriving respectively from [46] and [45], which exploit the rationale behind the attack proposed in Sect. 4.3.3.

4.4.1 Proportional Corruption

A simple attack mitigation mechanism alters the value of the counter by adding a certain *corruption level* in order to hide the statistical properties of the PMC, according to:

$$PMC' = cf(PMC) = PMC + cl \tag{4.8}$$

where PMC' is the corrupted PMC reading, PMC is the correct PMC reading, and cl is the *corruption level*.

The criteria we propose to select the corruption level is to relate it to the PMC distribution of the AES encryption service and of the different tasks executed in the system. In order to do that, we compute what we call the *Task Distribution Window* (TDW) of each task defined as:

$$TDW_{task} = \frac{|\mu_{task} - W_{TH_{task}}|}{2} \tag{4.9}$$

where μ_{task} is the average value of the selected PMC obtained when profiling the task (see Section 4.3.2) and $W_{TH_{task}}$ is the warning threshold of the task. The idea behind the task distribution window is to define a window of values that, in the average case, are able to poison the value of the PMC without moving a safe sample out of the safe region.

Once the TDW of each task has been computed, it is important to analyze how the different TDWs are distributed. As an example, Fig. 4.4 shows TDW_{AES} (red dashed line) and TDW of all tasks considered in our experimental setup for CCC (see Section 4.5). If a clear gap between TDW_{AES} and $min_{\forall task}(\text{TDW}_{\text{task}})$ exists (as in the case of Fig. 4.4), it can be used to generate the corruption in such a way

to significantly impact the AES timing (i.e., values higher than TDW_{AES}) but still safe for the other tasks (i.e., values lower than $min_{\forall task}(\text{TDW}_{\text{task}})$:

$$cl = U(0, s \times TDW_{AES}) \tag{4.10}$$

where U denotes that cl is sampled from a uniform distribution in the defined interval and s is a scaling factor that allows us to move the corruption level, thus trading-off its positive effect on hardening the attack with its negative effect on the timing violation detection.

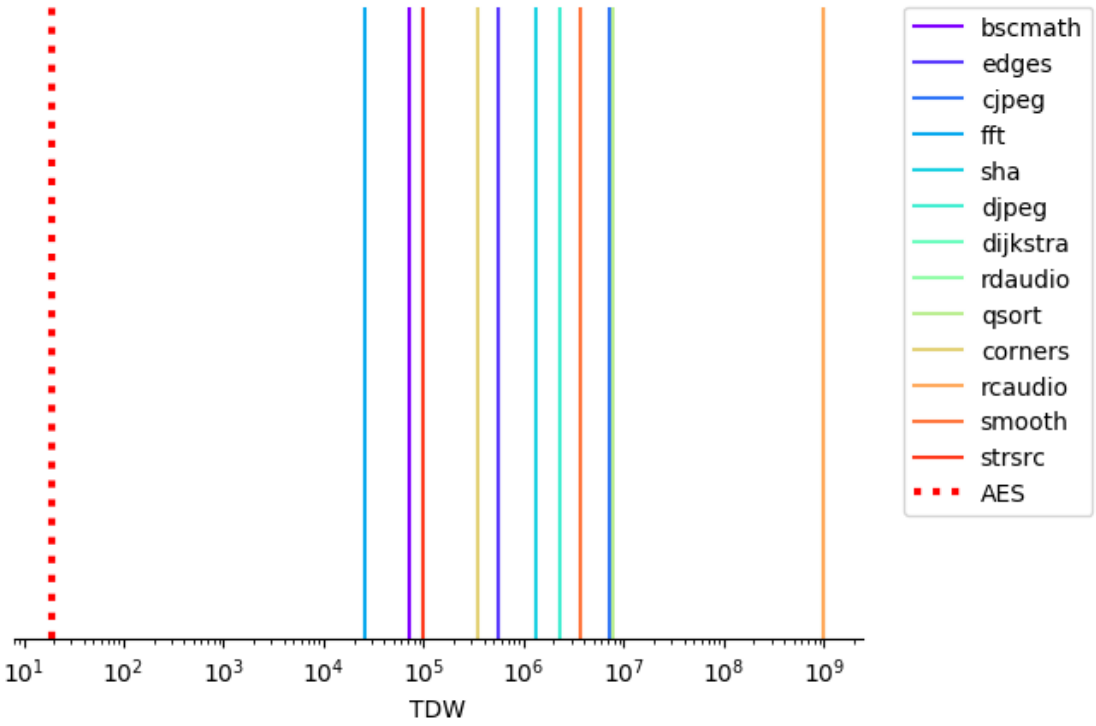

Figure 4.4: Tasks (MiBench) profiling for CCC. Vertical lines represent the position of TDW of the selected tasks.

As a result of this approach, every time a performance counter related to a user process is read, the corruption level changes. This consequently randomizes the differences between the PMC readings, which are at the base of the implementation of the attack described in Section 4.3.3.

4.4.2 Selective Corruption

The collision condition introduced in equation (4.6) tells us that samples with PMC values (i.e., CCC or DCM) significantly lower than the average value computed over all collected samples are the ones actually important to recover the key.

To obfuscate the properties of these samples, one option is to alter their values in such a way to violate the collision condition, leaving the remaining values unaffected by the poisoning.

To identify the values to alter, we exploit the CDF obtained profiling the PMCs from the encryption service. Fig. 4.5 reports the CDF computed profiling the CCC counter over 100,000 samples.

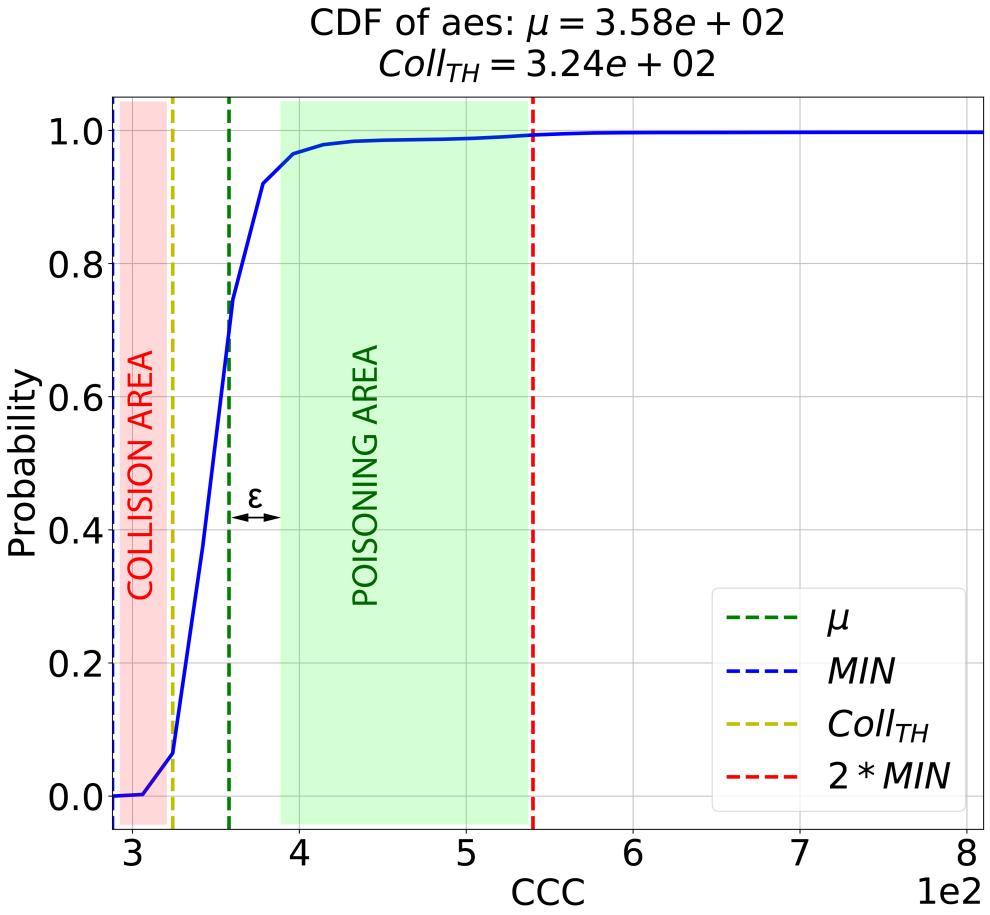

Figure 4.5: CDF of the CCC counter for the encryption service. Vertical lines represent the position of the selected thresholds. $Coll_{TH}$ is set here in order to identify the lowest 5% samples. Figure taken from [45] ©2019 IEEE.

The proposed solution is to set a threshold ($Coll_{TH}$) in such a way to identify the lowest x% samples of the CDF. Based on $Coll_{TH}$, it is possible to identify a *collision area* delimited by the lowest collected sample (MIN) and $Coll_{TH}$. Every sample falling in the collision area is moved to a random position of a different area called *poisoning area* (Fig. 4.5) by defining the corruption function of eq. (4.7) according to the following equation:

$$cf(PMC) = \begin{cases} PMC & (PMC < MIN) \vee (PMC > Coll_{TH}) \\ U(\mu + \varepsilon, ub) & MIN \leq PMC \leq Coll_{TH} \end{cases} \tag{4.11}$$

where PMC is the value of the counter, MIN is the lowest sample collected when profiling the encryption service and μ is the average value of the PMC over all samples collected during the profiling of the encryption service.

The upper bound of the poisoning area (ub) must not exceed two times the value of the minimum collected sample. This is motivated by the fact that, as described in [33], one of the techniques exploited by the attacker to optimize the attack is to discard samples higher than two times the minimum, considering them as outliers.

The lowest bound of the poisoning is instead defined by the ε parameter in order to create a guard band between the average (μ) and the beginning of the poisoning area. This is required since the effect of the corruption is to increase the average of the collected samples. The guard band can be empirically computed and must be large enough to guarantee that, even after the application of the poisoning, the corrupted samples are higher than the new average.

The decision of how many samples to corrupt (i.e., the selection of $Coll_{TH}$) and the size of the guard band (i.e., the selection of ε and ub) is a trade-off between security and amount of corruption. Higher values of $Coll_{TH}$ and ε increase the poisoning level and therefore the complexity of the attack (see Section 4.5). However, increased poisoning levels may have a negative effect on those safety techniques that rely on the PMC to detect anomalies. Therefore, the selection of these parameters must be carefully analyzed considering the interplay between safety and security as will be discussed in Section 4.5 where the different values of poisoning will be selected in relation to the safety technique presented in Section 4.3.2.

4.4.3 Observations

It is important to highlight that, a single corruption level is used for all tasks in the system in both attack mitigation techniques. This translates into a very simple implementation at the OS level. Moreover, this simplicity opens up a path for possible hardware implementations of both techniques that would relax some of the limitations currently imposed in the attack model. Nevertheless, this hardware implementation is out of the scope of this paper and will be considered in future extension of this research project.

As discussed in Section 4.3.3, the proposed techniques do not guarantee by proof the prevention of the considered attack. Differently, both techniques increases the complexity of the attack that, in this case, translates in the requirement of collecting an increased number of samples in order to discover the secret key. This is an important result. In general a careful design of a CPS under the security domain would supply a limited operational lifetime of the encryption key, as well as key

replacement policies. In this paper we do not consider how the different keys are generated or derived from previous keys, nor how they are distributed among the nodes of the CPS. Several techniques do exist in this domain [22, 263]. Instead we would like to discuss the impact of the proposed techniques on the lifetime of the key. Experimental data (see Section 4.5.2) show that the proposed approaches increase the number of samples required to perform a successful attack, especially when employing the mitigation technique of Sect. 4.4.2. Without considering the additional time required to analyze the samples (this can be carried out off-line with the support of high-performace computing facilities) the time required to collect additional samples has a positive impact on the possible lifetime of a key, thus reducing the overhead associated to the key generation and distribution among the nodes of the CPS.

4.5 Experimental Results

4.5.1 Experimental setup

To show the proposed approach at work, we evaluated both attack mitigation techniques on a sample computing node.

To account for the fact that CPSs cannot constantly update their hardware architecture, we selected for our experiments a board equipped with a powerful but relatively old Intel Core i7-720QM CPU (released in Sept. 2009). This processor is based on the Intel first generation Nehalem microarchitecture. It embeds 4 cores, each equipped with a 64KB L1 cache (32 KB L1 data and 32 KB L1 instruction) and a 256KB L2 cache. Finally, the processor implements a 6MB shared L3 cache.

To simulate the execution of different tasks we selected a set of 13 benchmarks from the MiBench benchmark suite [101] executed on a Linux operating system (kernel 3.11). These benchmarks represent typical algorithms implemented in several embedded systems sectors:

- *Automotive/Industrial*: susan-edges (edges), susan-corners (corners), susan-smoothing (smooth), quick sort (qsort), basic math tests (bscmath);
- *Consumer*: jpeg encoder (cjpeg), jpeg decoder (djpeg);
- *Office*: string search (strsrc);
- *Network and Security*: Dijkstra's algorithm (dijkstra), Secure Hash Algorithm (sha);
- *Telecommunications*: Fast Fourier Transform (fft), ADPCM encoder (rcaudio), ADPCM decoder (rdaudio).

Each task was profiled as described in Section 4.3.2 by collecting the value of the CCC and DCM counters over 100,000 executions. Input data for each task were taken from the "small" data set provided with the MiBench suite [101]. Profiling data has been collected separately for both techniques, on the same experimental setup.

The collected profiles were used to compute W_{TH}, C_{TH} and the average PMC value (μ) of each task for the two considered counters as reported in Table 4.1. Another set of profiles were used to compute W_{TH}, C_{TH} and the TDW, which is reported in Table 4.2. The confidence levels to compute the two thresholds were set to $C_W = 4\%$ and $C_C = 0.6\%$ and according to equation (4.3) $\alpha = 3$.

Table 4.1: Results of the profiling of the selected benchmarks employed in the Selective Corruption mitigation technique. This also includes the profiling of the encryption service (AES). Table taken from [45]. ©2019 IEEE.

	CCC counter			DCM counter		
Benchmark	W_{TH}	C_{TH}	μ	W_{TH}	C_{TH}	μ
AES	3.58e2	3.96e2	5.59e2	7.37e1	7.70e1	7.90e1
bscmath	5.35e6	5.50e6	6.21e6	1.69e1	2.30e1	2.60e1
cjpeg	1.20e7	2.64e7	2.95e7	3.31e4	3.33e4	3.36e4
corners	6.84e5	1.38e6	1.39e6	9.27e2	1.04e3	1.05e3
dijkstra	1.44e7	1.46e7	1.67e7	5.45e4	5.51e4	5.59e4
djpeg	3.38e6	8.01e6	8.11e6	6.73e3	6.96e3	7.23e3
edges	1.25e6	2.36e6	2.37e6	1.37e3	1.59e3	1.61e3
fft	3.54e5	4.04e5	4.32e5	3.78e3	4.10e3	4.31e3
qsort	1.70e7	3.27e7	3.28e7	4.18e5	4.19e5	4.21e5
rcaudio	2.15e9	4.12e9	4.27e9	1.84e9	3.54e9	3.66e9
rdaudio	2.15e9	4.12e9	4.27e9	1.80e9	3.46e9	3.58e9
sha	2.94e6	5.64e6	5.64e6	2.45e2	3.19e2	3.58e2
smooth	7.38e6	1.47e7	1.47e7	2.74e2	3.56e2	3.89e2
strsrc	1.26e5	3.24e5	3.24e5	1.49e2	1.75e2	2.05e2
synth01	2.69e2	3.96e2	5.33e2	4.84e1	7.70e1	7.80e1
synth02	2.88e2	3.96e2	5.36e2	6.38e1	7.70e1	7.80e1
synth03	3.35e2	3.96e2	5.34e2	5.58e1	7.70e1	7.80e1

For the CCC counter, Table 4.2 shows a clear gap between $\text{TDW}_{\text{AES}} \simeq 1.92e1$ and $\text{TDW}_{\text{fft}} = 2.53e4$ (the smallest TDW of the considered tasks). According to equation (4.10) it is therefore possible to explore a large range of scaling factors when poisoning this counter. A completely different situation arises when considering the DCM counter. This PMC generates less TDW variations and no clear gap is visible between $\text{TDW}_{\text{AES}} \simeq 1.64$ and $\text{TDW}_{\text{bscmath}} \simeq 3.05$ (the smallest TDW of the considered tasks). This makes it difficult to find a proper scaling factor not

Table 4.2: Results of the profiling of the selected benchmarks employed in the Proportional Corruption mitigation technique. This also includes the profiling of the encryption service (AES).

	CCC counter			DCM counter		
Benchmark	W_{TH}	C_{TH}	TDW	W_{TH}	C_{TH}	TDW
AES	3.96e2	5.59e2	1.92e1	7.70e1	7.90e1	1.64e0
bscmath	5.50e6	6.21e6	7.31e4	2.30e1	2.60e1	3.05e0
cjpeg	2.64e7	2.95e7	7.22e6	3.33e4	3.36e4	1.30e2
corners	1.38e6	1.39e6	3.46e5	1.04e3	1.05e3	5.47e1
dijkstra	1.46e7	1.67e7	9.73e4	5.51e4	5.59e4	3.11e2
djepg	8.01e6	8.11e6	2.32e6	6.96e3	7.23e3	1.14e2
edges	2.36e6	2.37e7	5.57e5	1.59e3	1.61e3	1.06e2
fft	4.04e5	4.32e5	2.53e4	4.10e3	4.31e3	1.62e2
qsort	3.27e7	3.28e7	7.88e6	4.19e5	4.21e5	1.55e2
rcaudio	4.12e9	4.27e9	9.88e8	3.54e9	3.66e9	8.48e8
rdaudio	4.12e9	4.27e9	9.88e8	3.46e9	3.58e9	8.29e8
sha	5.64e6	5.64e6	1.35e6	3.19e2	3.58e2	3.69e1
smooth	1.47e7	1.47e7	3.67e6	3.56e2	3.89e2	4.12e1
strsrc	3.24e5	3.24e5	9.93e4	1.75e2	2.05e2	1.30e1

affecting the safety task. The only benchmarks that show a clear separation in this case are rcaudio and rdaudio. These tasks are the most computation intensive of the selected benchmarks. This suggests that DCM is probably better suited only when the node runs this type of computation intensive tasks. The consequences of this will be better discussed in the next sections.

Also the CCC counter in Table 4.1 shows a clear separation of several orders of magnitude between W_{TH}, C_{TH} and μ of the encryption service (AES) and the ones of all other benchmarks. Therefore, according to equation (4.11) and as will be analyzed in Section 4.5.3 the poisoning technique should have minimum impact on the monitoring capability of the safety task. Even if the gap is reduced for some benchmarks, the same separation can be observed when considering the CCC counter.

In order to better analyze how the safety task could be impacted by the presented poisoning technique for tasks with PMC profiles similar to the ones of the encryption service, a set of three synthetic profiles (synth01, synth02, and synth03 in Table 4.1) was generated. These profiles contain random PMC values with a distribution that resembles the one of the encryption service.

To perform the attack, we adapted the C code presented in [33] available on the repository [32]. For each PMC, we collected 3 sets of samples on the node under attack. Each set contains 2^{30} = 1,073,741,824 samples, with the related ciphertext of 16 byte. The samples were transferred to a remote workstation equipped with

an Intel XEON E5-2680 2.70GHZ to be processed and attacked. For each set, the attack consisted in corrupting the samples, according to the techniques of Sect. 4.4.1 and Sect. 4.4.2, and performing the attack. For each set we performed 3 repetitions of corruption and attack.

The proportional corruption altered the samples with increasing scaling factors.

The selective corruption technique was evaluated considering six different values of $Coll_{TH}$, each defined in order to identify the lowest x% samples

$$x \in \{5\%, 7.5\%, 10\%, 12.5\%, 15\%\} \tag{4.12}$$

obtained when profiling the encryption service (see Section 4.4). For this purpose, the encryption service was profiled collecting 100,000 samples. To define the poisoning area taking into account the considered safety technique, we selected $\varepsilon = W_{TH}^{AES} - \mu^{AES}$ (see equation(4.11) and Table 4.1). This value is by construction large enough to guarantee a guard band as requested by our methodology. Moreover, to explore the impact of the size of the poisoning area on the effectiveness of the proposed technique, we evaluated different values for the upper bound of this area:

$$ub \in \{\frac{2 \cdot MIN}{16}, \frac{2 \cdot MIN}{8}, \frac{2 \cdot MIN}{4}, \frac{2 \cdot MIN}{2}, 2 \cdot MIN\}. \tag{4.13}$$

To analyze the behavior of the safety task, we profiled both the CCC and the DCM counters over 100,000 executions of each benchmark. After the profiling was completed we generated 1,000 traces for each benchmark (each of them composed of 100,000 samples) applying the same corruption levels used to perform the attack. All samples of each trace were classified according to the technique discussed in Section 4.3.2.

4.5.2 Attack mitigation results

The attack complexity can be measured for an attacker as the number of PMC samples needed to retrieve the encryption key.

Figs 4.7 and 4.6 show the number of samples required to perform the attack for the CCC counter respectively for the proportional and the selective corruption techniques considered.

The number of samples required to retrieve the encryption key without any PMC poisoning is on average $\sim$ 135,000,000 samples. This defines the baseline of the attack.

Fig. 4.7 shows that even the smallest alteration ($s = 1$) improves this complexity, at least doubling the required samples. Moreover, an increasing trend is maintained for increasing scaling factors. It is important to note that for $s = 3$ the attack failed 3 times out of the 9 attempts, and for $s = 4$ it failed 4 times. These failures cannot be numerically reported in the graph but are an indication of the increased

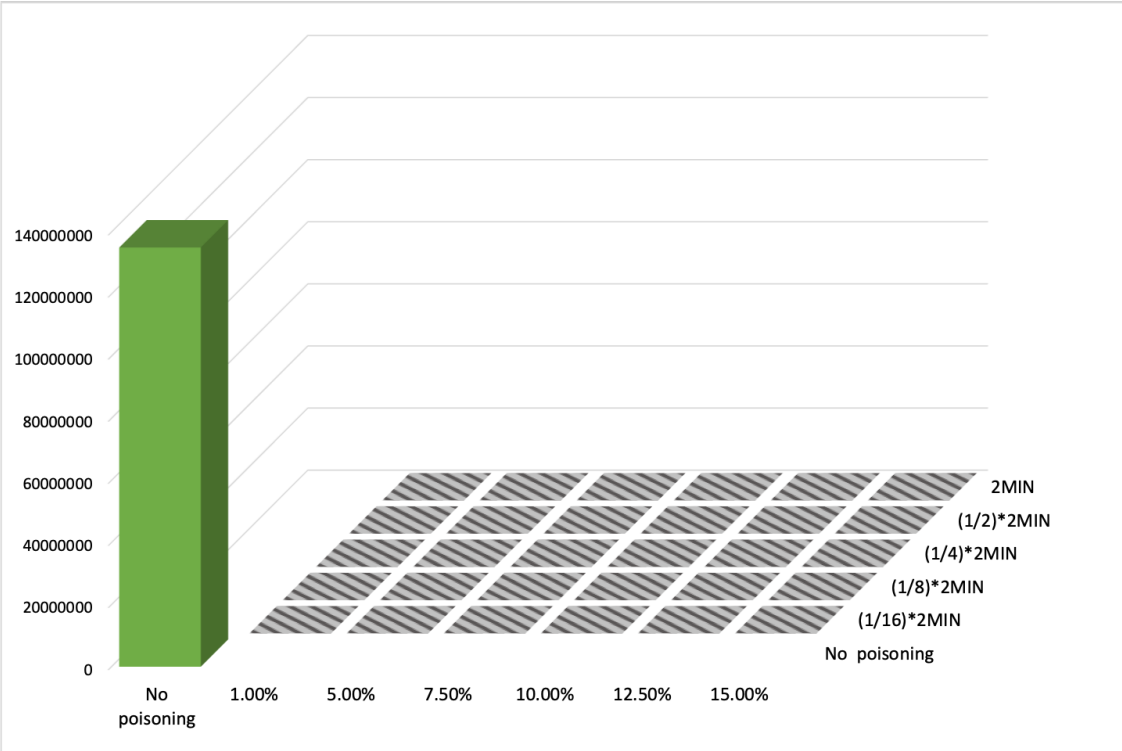

Figure 4.6: Number of samples to perform the attack for the CCC counter. Results obtained using the poisoning technique presented in Sect. 4.4.2. Gray spots indicated the failure of the attack. Figure taken from [45]. ©2019 IEEE.

complexity of the attack. With values of s from 5 to 10 the attack systematically failed.

Fig. 4.6 shows that, regardless of the size of the collision (x axis) and poisoning area (y axis), the attack always fails after analyzing 2^{30} samples when applying the proposed attack mitigation technique. This is a significant improvement with respect to results obtained applying the technique proposed in [46] (Fig. 4.7). In this last case, the attacker is in general able to recover the key with less samples while introducing a higher level of corruption of the counter.

While it is impossible to prove that for an increased number of samples the attack would not succeed, it is important to highlight that the amount of data the attacker has to collect passes from ∼2.5GB for the baseline up to ∼20GB when 2^{30} samples are collected. In a real scenario these data must be collected in a stealthy way, without draining all the resources of the system. Therefore, this could require a significant amount of time that, coupled with the use of key replacement keys (see Section 4.4.3), may further help counteracting the attacker.

Figs 4.9 and 4.8 report a similar analysis for the DCM counter. The attack against the unaltered set of samples requires on average 524,288 samples to discover

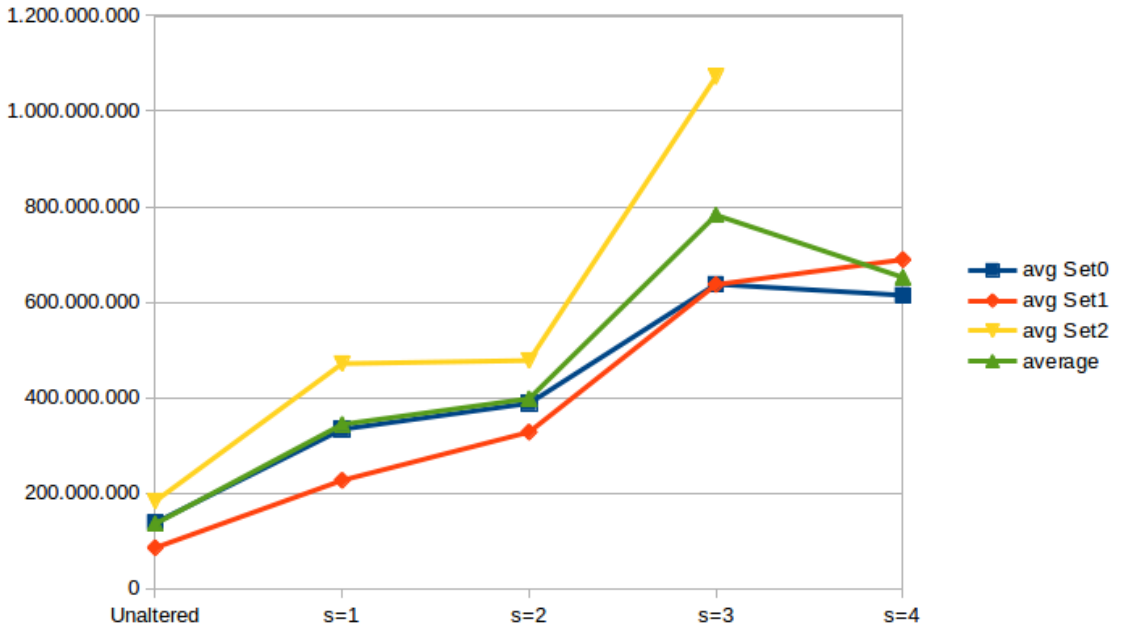

Figure 4.7: Number of samples to perform the attack for the CCC counter. Results obtained by applying the poisoning technique presented in Sect. 4.4.1, using different scaling factors. Blue, orange and yellow data report the average values of the 3 repetitions of the experiment for each of the 3 considered sets of samples, while green values report the global average. Figure taken from [45]. ©2019 IEEE.

the key. This value is significantly lower than the one required to perform the attack using the CCC counter.

When looking at samples collected for the two considered PMCs (i.e., CCC and DCM), we noticed, as expected, that DCM samples are more stables (i.e., they have a lower variance). Differently, as discussed in Section 4.3.3, the CCC counter is affected by several complex HW/SW interactions that are in general hard to predict. These interactions introduce variations in the timing behavior of the application across different executions, creating a higher variance in the observed values of the CCC timer and reducing its correlation with the number of cache misses (see Section 4.3.3). This motivates the reduced number of samples required to perform the attack using DCM. The higher stability of this PMC makes it difficult to protect using the technique proposed in [46]. Even with very high scaling factors (Fig. 4.9), the poisoning technique proposed in [46] is unable to introduce sufficient improvements from the security point of view. Moreover, this high poisoning would introduce an unacceptable corruption from the safety point of view. More in details, we observed that with this technique the altered samples maintain a similar allocation with respect to the average value, i.e., the majority of the original samples lower than the average remain lower than the average even

after the alteration. They therefore retain the information exploited by the attacker to recover the key.

Differently, the technique proposed in Sect. 4.4.2 (Fig. 4.8) is able to efficiently protect also the DCM counter. As for the CCC counter, the attack always fails with the only exception of the extreme cases in which both the collision area and the poisoning area are strongly reduced. Nevertheless, even in this cases we observe significant improvements in the number of samples required to perform the attack.

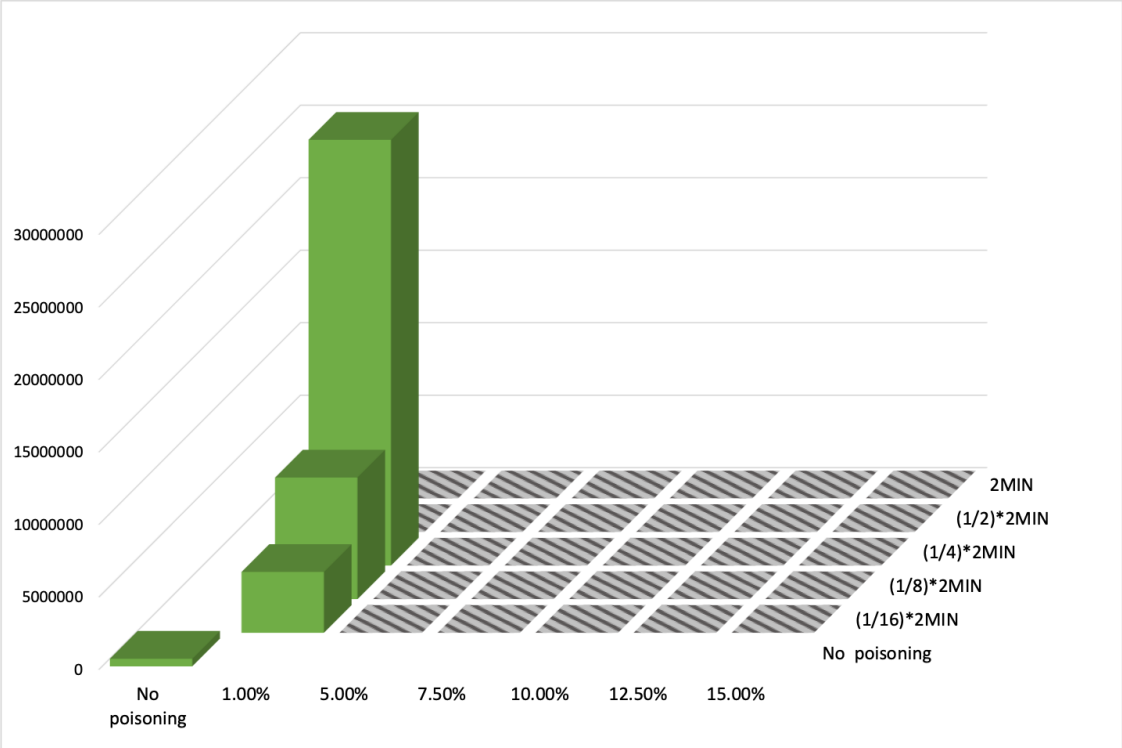

Figure 4.8: Number of samples to perform the attack for the DCM counter. Results obtained using the poisoning technique presented in Sect. 4.4.2. Gray spots indicated the failure of the attack. Figure taken from [45]. ©2019 IEEE.

To summarize, employing the selective corruption as attack mitigation technique, provides optimal results from the security perspective, considering both DCM and CCC counters.

4.5.3 Safety results

The proportional corruption technique of Sect. 4.4.1 has been employed. Fig. 4.10 shows, for each task, the number of executions that fall in the different safety states introduced in Section 4.3.2: safe direct (Fig. 4.10a), critical direct (Fig. 4.10b), safe warning (Fig. 4.10c), and critical warning (Fig. 4.10d). Results are provided for

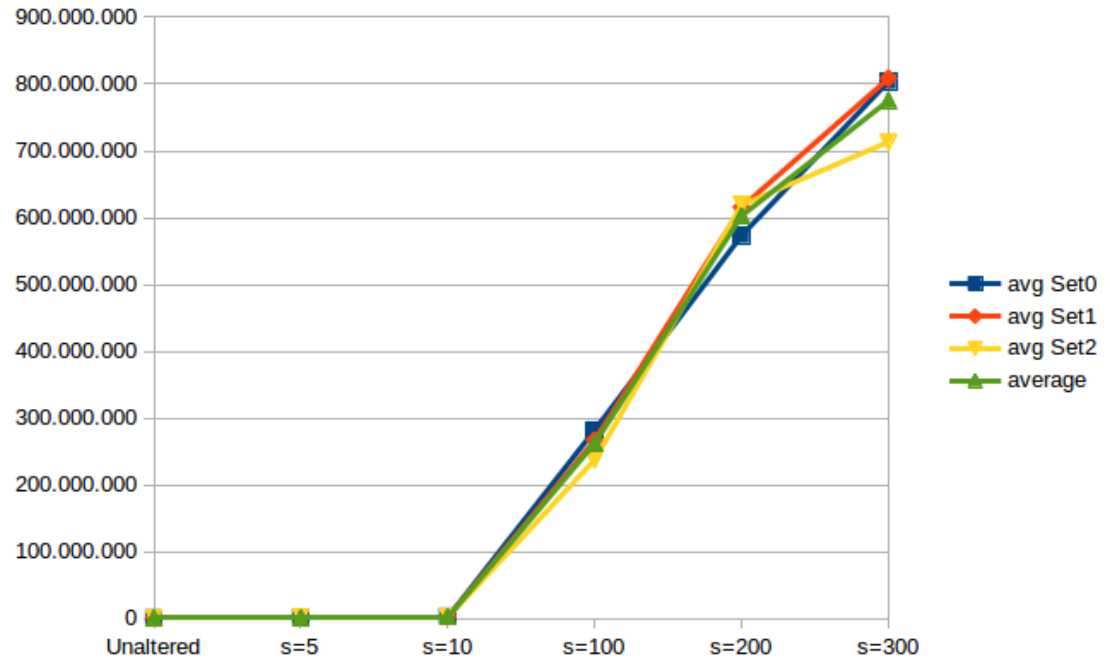

Figure 4.9: Number of samples to perform the attack for the DCM counter. Results obtained using the poisoning technique presented in Sect. 4.4.1 using different scaling factors. Blue, orange and yellow data report the average values of the 3 repetitions of the experiment for each of the 3 considered sets of samples, while green values reports the global average. Figure taken from [45]. ©2019 IEEE.

different corruption levels of the CCC counter, averaging the results over the 1,000 repetitions of the analysis.

The impact of the alteration of the CCC counter on the safety of the benchmarks is negligible. This means that the number of critical executions does not grow significantly (Fig. 4.10b and 4.10d). More specifically, the increment of the critical direct executions is below 1.75% (Fig. 4.10b), and the same happens for the critical warning executions (Figure 4.10d) apart from *fft* whose growth is lower than 40%. The reason of the behavior of *fft* is that a small set of executions is very close to the thresholds. Therefor, a minimum amount of corruption changes their state. This is also supported by the decrement of executions classified as safe directly (Figure 4.10a) and the increment of safe warning executions (Figure 4.10c).

Fig. 4.11 reports a similar analysis considering the DCM counter. As expected, given the TDW distribution for this counter (see Section 4.5), overall the safety detection capability is considerably compromised.

However, observing accurately Fig. 4.11b, reporting the number of the executions classified critical direct, the analyzed benchmarks can be grouped into two groups: one of them heavily influenced by the corruption and another which is

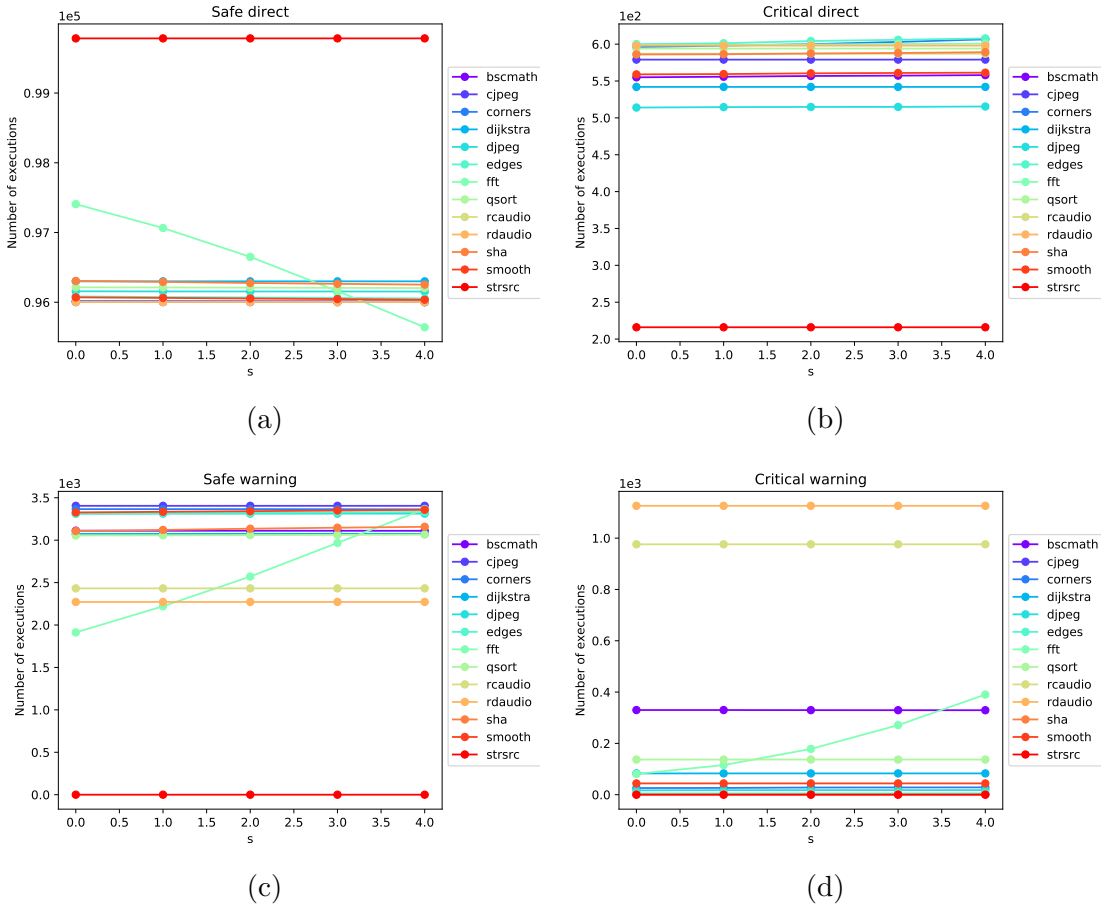

Figure 4.10: Impact of the CCC corruption on the safety state classification of each benchmark

not particularly affected. *Bscmath*, *corners*, *edges*, *sha*, *smooth* and *stringsearch* have a growth more than 1,000% and belong to the first group while the others to the second. As expected, this distinction is highly correlated to the TDW of the benchmarks, the smaller is the TDW, the higher is the impact of the corruption. A clear trend for the warning increment cannot be instead identified, as shown in Fig. 4.11c and Fig. 4.11d.

This confirms that the DCM counter is more vulnerable to attacks and therefore requires special attention when used. From the security perspective it requires significantly higher corruption levels that can be tolerated from the safety perspective only if the considered tasks have a TDW several orders of magnitude higher than TDW_{AES} (i.e., complex tasks). This is not the case for several benchmarks considered in our experimental setup.

Figs. 4.12 and 4.13 show, for each task, the number of executions that fall

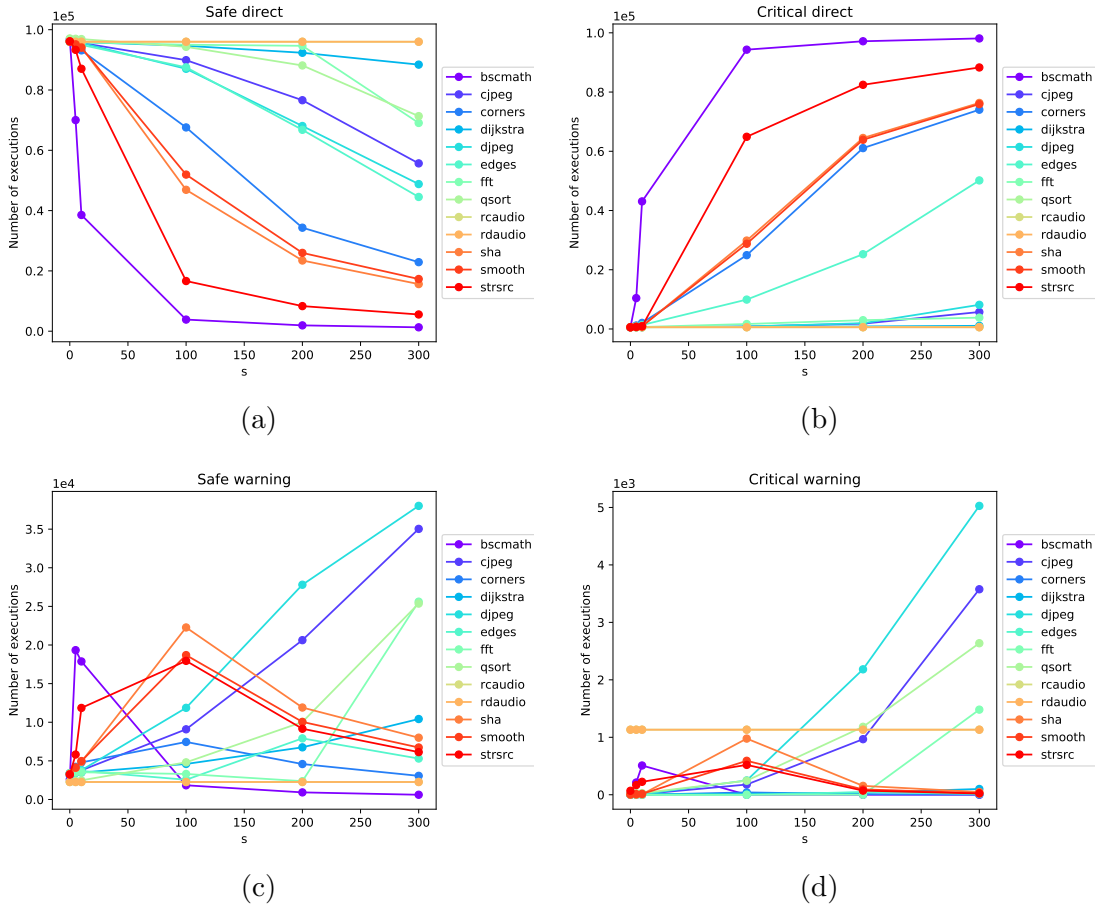

Figure 4.11: Impact of the DCM corruption on the safety state classification of each benchmark

in the different safety states introduced in Section 4.3.2 for the CCC and DCM counter, respectively. Subfigures are associated to the different safety states: safe direct (Figs. 4.12a and 4.13a), critical direct (Figs. 4.12b and 4.13b), safe warning (Figs. 4.12c and 4.13c), and critical warning (Figs. 4.12d and 4.13d). Results are provided for different corruption levels of the CCC and DCM counters, averaging the results over 1,000 repetitions of the analysis. The alteration has been performed according to the selective corruptio technique of Sect. 4.4.2

Looking at the figures it is clear that the MiBench tasks are not impacted by the alteration of both the CCC and DCM counters for all considered percentage sizes of the collision area (x axis). For all cases we used the largest poisoning area (i.e., $ub = 2 \cdot MIN$). This is favored by the fact that W_{TH} and C_{TH} of all these tasks are significantly different when compared to the considered poisoning windows and is a significant improvement with respect to the results presented in [46].

Some changes can be observed only when looking at the synthetic benchmarks.

Looking at the CCC counter, in the worst case (synth02) the number of safe direct classifications drops from 96,018 to 77,035 generating 18,985 potential fault positives. This is partially compensated by an increment of 3,884 safe warning cases. Nevertheless, the remaining 15,101 cases out of the 100,000 total executions represent false positives that must be handled. This impacts the performance of the system due to an increased number of recovery actions required during in field operations. A similar analysis can be carried out for the DCM counter. By construction, no false negatives can be introduced by the proposed technique, since the corruption is always introduced as an additive positive factor.

Even if this may look a negative result, it is important to recall that the profiles of the synthetic benchmarks have been specifically generated to resemble those of the encryption service. Therefore they represent an extreme worst case situations difficult to encounter in real applications. These results must be only considered as examples to better show the critical aspects that should be carefully taken into account when designing a CPS in which both safety and security must be guaranteed as described in Sect. 4.3.

4.6 Conclusions

This chapter studied the interplay of two challenging aspects of the design of a CPS: safety and security. It focused on the role that the PMCs have when implementing mechanisms able to enhance the safety of the system and, on the other hand, the risks they introduce when looking at the security of the system. Starting from the example of a PMC based safety mechanism, and from the implementation of a security attack, here is proposed an attack mitigation strategy. Two different PMC were analyzed and an extensive experimental campaign shows the effectiveness of the proposed attack mitigation technique for both considered counters. This

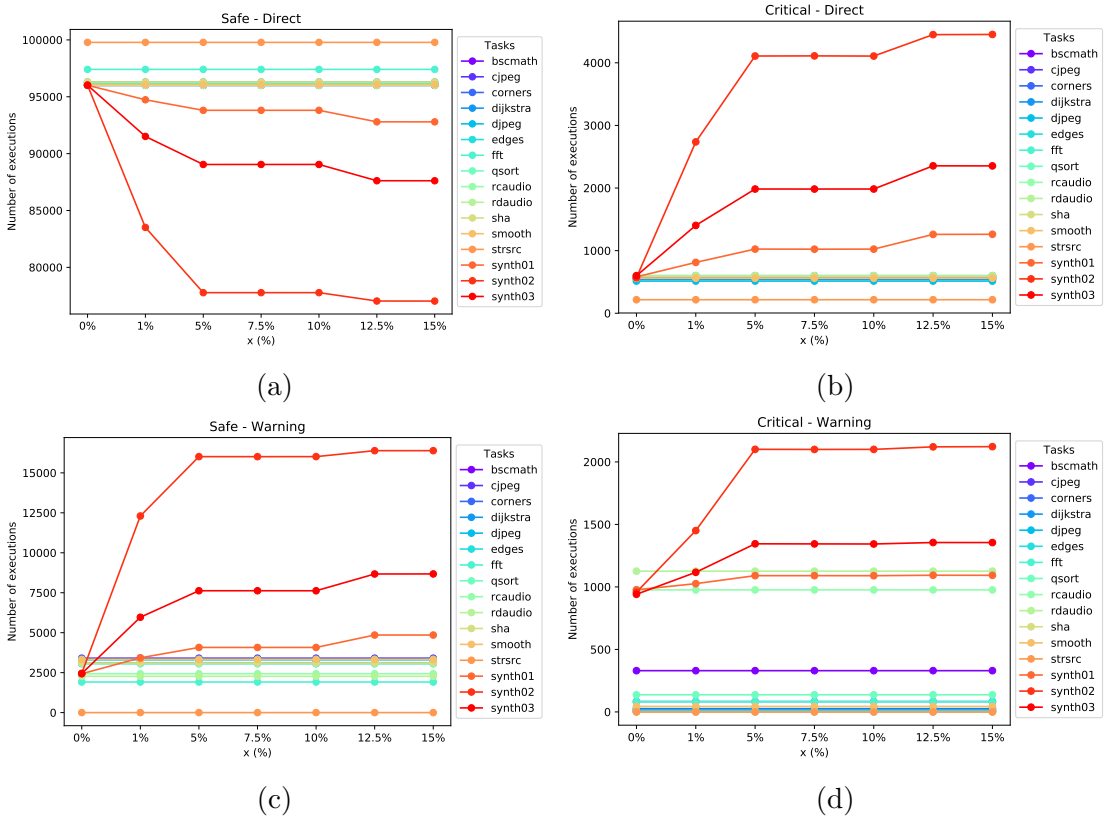

Figure 4.12: Impact of the CCC corruption on the safety state classification of each benchmark. Figure taken from [45]. ©2019 IEEE.

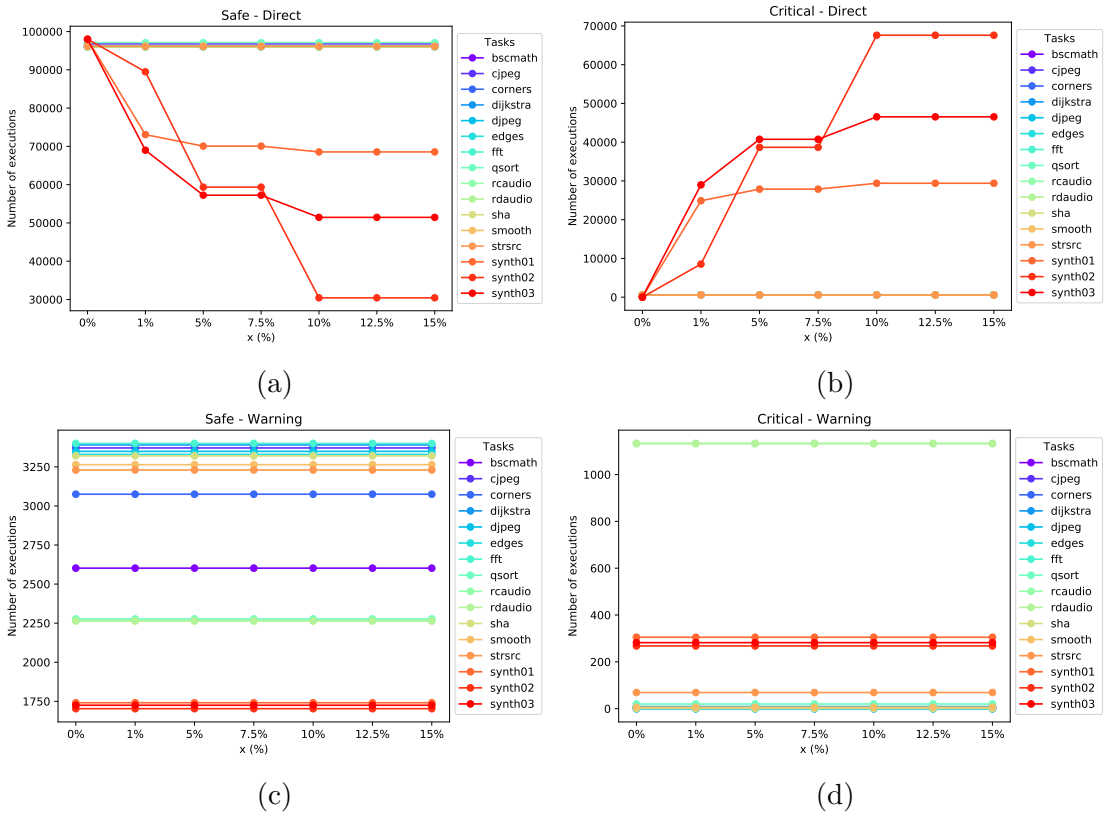

Figure 4.13: Impact of the DCM corruption on the safety state classification of each benchmark. Figure taken from [45]. ©2019 IEEE.

provides interesting suggestions on how a designer should decide which PMC can be securely exposed to the application software.

Chapter 5

FPGA Attacks

Reconfigurable platforms are employed in Critical Infrastructures to exploit addional flexibility and acceleration capabilities. However, FPGA-based systems are vulnerable to attacks targeting the design to be deployed in-the-field. This chapter describes scenarios in which mobile heterogeneous devices exploit the computational capabilites of the reconfigurable embedded hardware. The scenarios described deals with attacks targeting the confidentiality, integrity and availability of the bitstream loaded onto the FPGA and deals with local and remote attackers. The remainder of this chapter is an extended and revised version of the publications [47, 48].

5.1 Introduction

The current mobile application market is constantly changing due to the presence of new devices and platforms which emerge quite frequently [10]. The changes affect the business environment creating a demand but also an opportunity for the rapid introduction of new technological solutions. Players in the growing business landscape cannot ignore this opportunity, since mobile technology will eventually have a role in most digital products and services.

Several well-known companies participate in this scenario: the first to define the market was Apple which launched, in 2007, the iPhone and later, in 2008, its distribution platform App Store. Subsequently, other players and device manufacturers joined the mobile application market with their devices, operating systems and software distribution platforms. Currently, the distribution of mobile applications spans over 300 application stores worldwide, including device manufacturers, platform providers, mobile operators. Well-known examples are Google Play Store (previously known as Android Market), Apple App Store, Windows Phone Store, Opera Mobile Store, etc. [71]. Sales, finalized through a payment gateway, and downloads of mobile apps are skyrocketing too. According to a recent survey by iResearch, in 2018, global mobile app revenues amounted to over 365 billion U.S.

dollars. In 2023, mobile apps are projected to generate more than 935 billion U.S. dollars in revenues via paid downloads from app stores and in-app advertising [121].

Several motivations are at the base of the continuous growth of this phenomenon. Certainly, the hardware improvement is the key factor that keeps pushing applications toward new levels of pervasiveness, which allows an ever increasing computational power of mobile platforms. Second, heterogeneous computing has been the leading technology that allowed moving towards new generations of mobile devices. For example, combining multi-core processors with Graphics Processing Units (GPUs) and other types of hardware accelerators is having a huge impact on device performance. Therefore, System-on-Chip developers are increasingly looking at alternative architectural solutions to increase the computational power, while optimizing additional parameters such as power consumption.

In this landscape, reconfigurable computing is a promising solution. As shown in [83, 223], programmable system platforms embedding Field-Programmable Gate-Arrays (FPGAs) might be a valuable solution where frequent and remote upgrades are necessary, thus also for embedded applications. Moreover, some FPGA-based systems provide Dynamic Partial Re-configuration (DPR), which allows the run-time update of selected portions of an FPGA without disrupting the rest of the system. DPR allows the creation of new application scenarios [8, 77, 65] and it has already been used in the mobile application market [72, 222, 186]. Hardware vendors are responsible for manufacturing the FPGA devices and sell them to their customers or retailers. In FPGAs it is possible to (re)configure logic resources by controlling the interconnection among different logic gates. The hardware logic blocks implementing specific functions compose an Intellectual Property (IP) soft IP core. A soft IP core is an independent and reusable module that can be instantiated in the reconfigurable fabric. One or more soft IP cores are described by a *bitstream* file, which configures the FPGA (or just a portion, if DPR is supported). The project of a soft IP core is defined by the system designer. The exploitation of reconfigurable hardware enables a new mobile application scenario, where a reconfigurable device can be programmed at run-time to assist the execution of a software application by means of application-specific computational cores. The applications can employ hardware capabilities on-demand using ad-hoc computational resources optimized for the various system's aspects (e.g., power consumption of the whole system). Mobile applications like games, audio/video processing, secure communications are good candidates to benefit from providing application-specific hardware acceleration cores deployed together with the software application.

However, this new application paradigm opens up concerns in the security domain. As an example, an adversary that is able to intercept a bitstream of an hardware Intellectual Property can try to extract sensitive information or steal the property, thus leading to IP infringements. Also disclosure to the public domain or unauthorized sales to earn unfair profit are possible, thus violating the confidentiality of the hardware block. Considering the hardware, an attacker may also produce

intentional alterations to the hardware block by injecting malicious code that may either corrupt the correct behavior of the software application or compromise the whole end-users system, e.g., by introducing security flaws which could later be exploited.

Therefore, the hardware description of soft IP cores should never be sent across unprotected network links. FPGA manufacturers have already provided techniques, such as bitstream encryption and authentication, which try to address these issues. Indeed, they fit a simple scenario where the application developer and designer is the only entity entrusted to produce reconfigurable hardware descriptions to be delivered to a remote system. Certainly, they don't fit the current very dynamic and heterogeneous scenario of the mobile application market. Moreover, these methods may require an excessive involvement and trust in the manufacturers, which usually embed secrets in their hardware that can eventually be exploited. Even worse, it requires to trust the whole manufacturers' supply chain as well, which has become an issue with the delocalization of the production.

In this work, we tackle a much more challenging and realistic situation that involves several independent parties participating in the development and distribution of applications to be executed on heterogeneous mobile platforms embedding an FPGA as reconfigurable hardware. These parties include: the end users, the application stores, the software providers and the reconfigurable hardware vendors. In this scenario, a single reconfigurable hardware resource can be shared by several applications from different vendors with guarantees on the integrity and confidentiality of the provided hardware cores.

This chapter addresses the security threats introduced by two types of adversaries: (i) remote adversaries acting on the communication channels between the application providers and the devices (Man-in-the-Middle), and (ii) local adversaries with physical access to the system (Man-at-the-End).

We describe the security services for the envisioned infrastructure by taking into account three aspects: (i) the hardware resources and the system architecture to implement the required security primitives, (ii) the high-level software infrastructure needed to implement the required communication protocols, and (iii) the high-level entrusting policies required among the involved entities.

Instead of resorting to ad-hoc technology to tackle the adversaries and the security issues, we exploit the idea of another mainstream initiative, i.e., the Direct Anonymous Attestation (DAA) protocol [35]. Currently, the DAA protocol has been standardized by the Trusted Computing Group (TCG) [235] and it is supported in ad-hoc chips like the Trusted Platform Module (TPM). Even if we do not necessarily propose the use of TCG-specified TPM chips, we follow the progresses in this field employing reconfigurable hardware to achieve the same features. This also means that there will be more people researching for flaws, as the impact of attacks becomes greater thus also the impact of publication of such attacks is more likely to reach a higher visibility.

5.2 Assumptions, models and requirements

This section presents our model by describing the involved actors and the security and functional requirements to be satisfied. Moreover, it characterizes the two scenarios we assume in this thesis, which depict very common situations for the deployment of mobile applications.

The *application* is the object to be securely exchanged among all involved parties. As shown in Fig. 5.1, in this thesis we consider applications composed of two portions: (i) the executable code (SW), and (ii) an *FPGA Bitstream file* (BS).

The application is executed on the End User Device, which embeds a general purpose CPU executing the SW and a FPGA-based Hardware Acceleration Platform (HAP). HAP contains the reconfigurable logic, i.e., the FPGA, and a microcontroller used to securely load and store a bitstream.

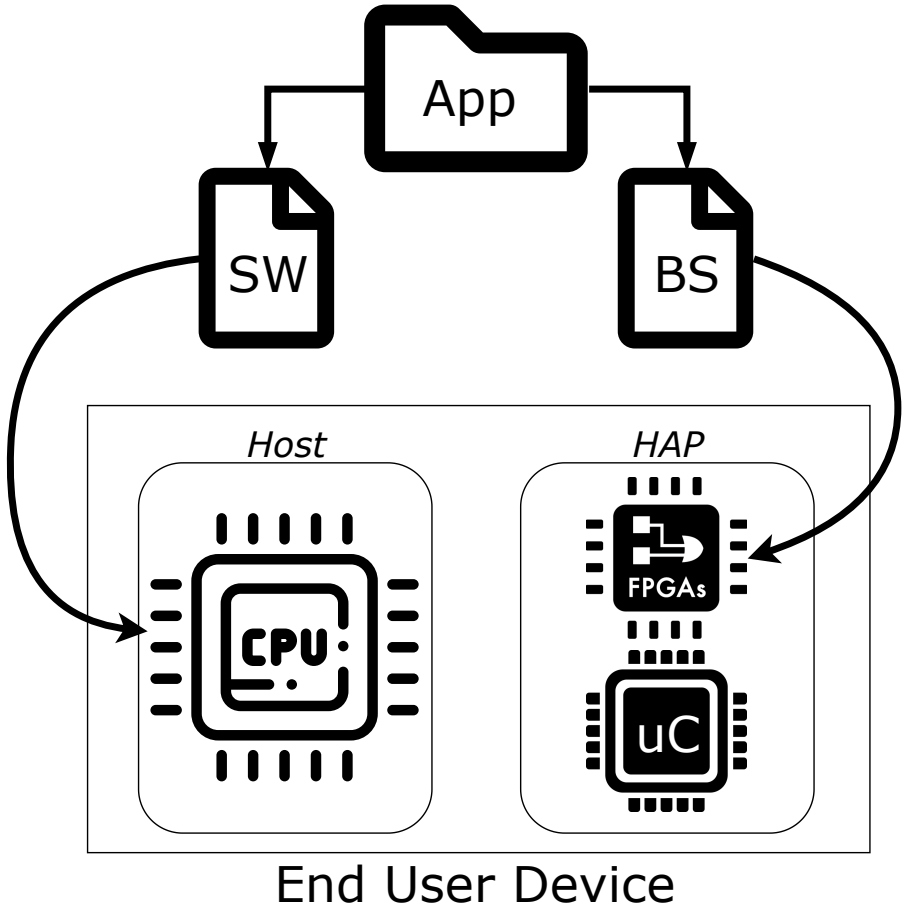

Figure 5.1: Mobile application paradigm considered mapped onto the (simplified) architecture of the End User Device.

The bitstream describes one or more soft IP cores that complement the application (e.g., required to improve its performance). These soft IP cores are dynamically configured in the FPGA every time the application is executed.

5.2.1 Actors

We model the considered scenarios following a common scheme for application deployment for the mobile application market. The actors and their high-level interactions are shown in Fig. 5.2.

We consider three main types of actors: (i) the software providers, (ii) the hardware vendors, and (iii) the end users.

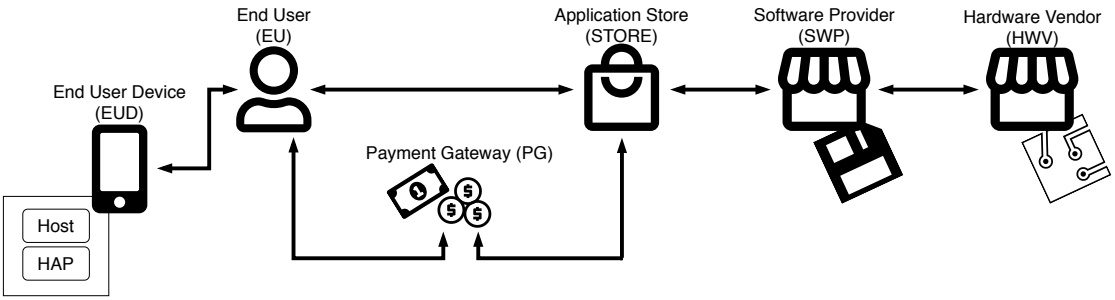

Figure 5.2: Actors involved in our scenarios showing a common mobile application deployment flow.

The Software Provider (SWP) is the entity that develops applications and sells them through several *Application Stores* (STORE) to reach a high number of potential customers.

The Hardware Vendor (HWV) is the entity responsible for designing and selling the Hardware Acceleration Platform (HAP) used by the applications to load the BS and it implements the soft IP cores developed and sold by SWPs. The Hardware Acceleration Platforms considered in this chapter are microprocessor-based Systems on Chips (SoCs) embedding state-of-the-art FPGAs featuring DPR and bitstream encryption mechanisms. The HWV has full knowledge of the hardware it produces, including secrets and security mechanisms it embeds (e.g., cryptographic keys). We assume that each hardware vendor has a publicly accessible service (e.g., a web server) used to offer services to the software providers and to the end users (e.g., prove hardware product authenticity, send firmware updates, etc.). It also provides publicly available information (e.g., list of hardware capabilities, APIs, list of known compromised devices).

The End User (EU) is the customer. The EU may be interested in buying applications that have been developed by different SWPs. He/she owns an End User Device (e.g., smartphones, tablets, PDAs, etc.) that embeds the HAP. The End User Device and HAP are two separate execution environments, therefore,

they are considered as two separate entities in our scenarios. However, the HAP does not directly access the network, all communications with the external world are mediated by the End User Device.

Together with the above three main actors, two more players are involved in the scenario we have considered in this chapter. The STORE acts as an interface between EUs and SWPs by collecting and making available to the EUs software applications developed by different SWPs. Every STORE has a publicly accessible service (e.g., a web server) and is connected to one or many payment gateways to allow customers to buy the software they want to purchase. Each STORE knows the identity of its users.

Finally, the *Payment Gateway* (PG) offers payment services for software application purchases (e.g., using credit cards), interfacing with SWPs and with EUs.

5.2.2 Security requirements

The high-level security objective of an SWP is to preserve its intellectual property. Moreover, an SWP aims at securing its applications, by preserving their authenticity, integrity and confidentiality. The EUs require to only execute authentic versions of the software they have bought.

We solely consider the security aspects of the hardware cores, while ensuring authenticity and integrity of executable code is out of the scope of this thesis. Interested readers may refer to the literature on this field for a better understanding of available methodologies for software protection [60, 166, 81, 23].

The security requirements to be fulfilled in order to guarantee the authenticity and the intellectual property of all hardware cores deployed are the following ones:

- *Bitstream confidentiality*: no other players but the SWP itself must be able to read a bitstream in an intelligible form (e.g., if the application bitstream files are encrypted, the bitstream should not be available in plaintext). Every SWP does not trust neither other SWPs nor the Applications Stores used for distributing applications. Moreover, the SWP does not trust the EUs, even those who have legitimately bought its application.

  We have considered two scenarios in this thesis. The first one (named Simple Scenario and presented in Section 5.3.1) considers a case where a trust relationship exists between the SWP and the HWVs (i.e., HWVs might read the bitstream in plaintext) because of legal contracts or Non Disclosure Agreements (NDAs). The second scenario, which tighten the confidentiality (named Full Scenario and presented in Section 5.3.2), relaxes this assumption, avoiding every trust relationship among all the parties involved.

- *Bitstream integrity and authenticity*: corrupted bitstream files must not be delivered to the EUs, or used to configure the FPGA. The types of corruption

to avoid are both accidental (e.g., transmission errors) and deriving from intentional malicious alterations. Moreover, the EUs must be ensured that the bitstream files they are loading in their HAP are genuine, that is, they have actually been developed by the SWP.

- *Legitimate End User*: only users who bought a legitimate copy of the software must be able to use the related bitstream files to configure the FPGA available in the HAP to assist the execution of the related software application. Nevertheless, in no case end users are allowed to access the bitstream in plaintext.

- *Legitimate HAP*: only owners of legitimate HAP can run applications into a user platform and load a BS into the FPGA.

- *Need-to-know restrictions*: only the parties that actually need a sensitive information must have access to it. This is of great importance especially for information concerning the EUs (e.g., the applications that they buy). This requirement is related to the privacy, however the need-to-know is subjective, thus it cannot be considered a privacy statement.

If any of these requirements is not met, the security of a bitstream is undermined and an adversary would be able to either read the bitstream and start reverse engineering or compromise the bitstream authenticity and integrity to inject malicious features, e.g., by opening security backdoors.

5.2.3 Attack model

We consider two types of attacks: IP attacks and integrity attacks.

When a *IP attack* is performed, an adversary tries to get access to an intelligible version of a bitstream. This attacks can be performed either on the HAP by tampering with external memory devices or over the network links the application traverses during their deployment. This type of attack aims at violating the confidentiality of the bitstream in order to make illegal copies of the soft IP cores.

When an *integrity attack* is performed, an adversary tries to compromise the integrity of a bitstream. This type of attack has two motivations: (1) a remote adversary may want to compromise the bitstream integrity to prevent the correct execution of the related software application (Denial of Service), (2) a malicious user may want to replace a bitstream file with a previous version (downgrade), for example to avoid security updates.

In this thesis we do not consider physical attacks that aim at damaging the HAP or make it not operable (i.e., hardware Denial Of Service attacks).

5.2.4 Adversary model and security assumptions

The attacks just described can be carried out by a malicious attacker. We consider two types of adversaries trying to break the requirements introduced in Section 5.2.2; remote and local adversaries. We assume that both local and remote adversaries are knowledgeable attackers. They have considerable knowledge of the system and device they are attacking and they own high technical skills. Moreover, they have access to sophisticated tools and instruments [2]. Nonetheless, they are associated to different threat models.

A *remote adversary* controls every possible network link involved in the proposed scenarios, i.e., he/she can perform Man-In-The-Middle (MITM) attacks. A Remote adversary uses techniques to intercept and understand the content of the messages exchanged between two arbitrary networking nodes. Once the attacker modifies the flow of messages, he can also make changes, delete and create completely new fake messages impersonating one of the communicating parties.

A *local adversary* is a malicious EU who has physical access to and full control of a End User Device. That is, he/she can perform Man-At-The-End (MATE) attacks. Local adversaries have no restriction on the tools and techniques to reverse-engineer and then to tamper with the application (e.g., debuggers, emulators). Thus the application cannot be trusted to store/embed secret data or routines. System libraries and general purpose libraries could be controlled by local adversaries, along with the operating system. In this case, to reach their goals attackers can use and alter system calls, the input/output subsystem, the network stack, the memory management subsystem and possibly others techniques. Therefore, the communication with the HAP mediated by software and drivers can be compromised by these adversaries, thus the data exchanged can be altered even if they are transmitted via secure channels that are safe against MITM attacks. The attacker also controls the platform hardware. Every memory location can be read and written, including the processor registers. The attacker also controls the program storage medium, as a consequence he can read and change any of the stored bits at any time. This means that nothing can be considered secure in the user's environment.

The only part of the user's platform that we will consider secure is the HAP, the stored information and the routines executed within the HAP are considered confidential. Indeed, we assume that, even if the HAP may be placed in an hostile environment, the reconfigurable hardware and the security blocks surrounding it are resistant to physical attacks (e.g., decamping the chip, side-channel attacks, etc.).

Finally, we assume that the local adversaries have no interest in performing Denial of Service attacks against their platforms and including HAP (e.g., by repeatedly sending invalid bitstreams to block its functioning).

Competing firms are potential local adversaries, since they can invest non-negligible resources trying to compromise the security requirements.

Moreover, we assume that attackers all work under the *infeasibility* hypothesis. The infeasibility is associated to the derived computational cost, impossible to sustain for an attacker, in order to decipher the secret information needed by cryptographic algorithms or to invert digest algorithms. In practice, attackers are unable to solve exponential problems of proper size in useful time.

Eventually, when we state that two peers communicate with a k-secure channel, we indicate that the peers perform strong authentication and agree on a symmetric key k that can then be used with data integrity and authentication algorithms and symmetric encryption algorithms to secure the exchanged data.

5.3 Protocols and secure information exchange

This section introduces our solution for the secure reconfigurable computing model presented in Section 5.2. In particular, we will introduce and analyze the information exchange and the required protocols that will be used to identify the required hardware structures. Two cases will be analyzed:

1. a simple scenario fulfilling a reduced set of security requirements but exploiting minimum hardware facilities, and

2. a complex scenario featuring trusted computing hardware fulfilling the full set of security requirements of Section 5.2.

In both scenarios, the target of the protocol is to define the operations to deploy a mobile application, composed of the software and its hardware counterpart, on the End User Device. However, the transfer among the involved parties has to be secured with respect to the considerations introduced in Section 5.2 and overcoming the limitations discussed in Sect. 5.1.

5.3.1 Case 1: Simple Scenario

In this scenario, a simplified version of the protocol is presented. This protocol satisfies a reduced set of security requirements, but benefits from simplicity and reduced hardware requirements.

Fig. 5.3 shows a possible workflow for the deployment and execution of an application onto a device embedding reconfigurable computing resources.

In this simple scenario, we assume that there is a trust relationship between the SWP and the HWV. Moreover, together with the security requirements and assumptions presented in Section 5.2, the following realistic functional assumptions are considered:

- The EU is able to access the store (i.e., he has an account on the STORE);

- The EU has a credit card or any another equivalent payment system compatible with the store;
- The STORE has existing agreements with one or more payment systems;
- The HAP is identified by a unique code (e.g., serial number), defined here as id_{HAP} and stores a secret cryptographic key (K_{HAP}), both known by the HWV. The key is not accessible from the outside of the HAP;
- all involved entities are able to create secure channels (e.g., through SSL/TLS protocols) satisfying confidentiality, data integrity and authentication, and peer authentication using an agreed symmetric key;
- the SWP knows the HAP manufacturers (i.e., HWVs) and can access to their services.

The following protocol presents the steps to be performed in order to buy an application and securely obtain the associated bitstream (see Fig. 5.3).

1. (*The* EU *buys the application*) The EU browses the STORE and decides to buy a mobile application, one that is composed by SW and BS, developed by a SWP. As usual in the web market era, the STORE redirects the user on the PG to complete the purchase. The PG notifies the STORE if the payment transaction terminates successfully. The STORE starts the procedure to obtain the requested BS to send to the client. The actual steps performed may change depending on the information exchanged between the STORE and the PG, thus this is not reported in Fig. 5.3.

2. (*The* EU *sends the* STORE *its data*) The first interaction reported in Fig. 5.3 involves the EU and the STORE. The EU sends (from its device platform) the information needed to identify its HAP, i.e., id_{HAP}. Since the communication involves sensitive data, the End User Device and the STORE use a secure channel that ensures confidentiality, data integrity and authentication by means of an agreed symmetric key K_{CS}[1]. The freshness is guaranteed by using a random number r_C. In Fig. 5.3, the symbols r indicate random numbers. The STORE then sends back an acknowledgement to the Client in the secure channel[2].

[1]To simplify the presentation, we indicate that K_{CS} is used both for symmetric encryption and to compute a Message Authentication Code (MAC), even if K_{CS} is better used as a master key to derive different keys (as actually done by TLS). Also note that an authenticated encryption primitive could have been used instead of a MAC.

[2]The acknowledgments are sent to confirm the correct data receipt after all the interactions. Nonetheless, they are not explicitly reported in the text to ease the reading.

3. (*The* STORE *notifies the* SWP) The second interaction reports the STORE that notifies the SWP that a new customer bought the application (thus also its BS). The STORE forwards information about the HAP of the client who bought the software. Again, the communication involves sensitive data (the $\mathrm{id_{HAP}}$) thus the STORE and the SWP communicate by using a secure channel that ensures confidentiality, data integrity and authentication by means of an agreed symmetric key K_{SS}.

4. (*The* SWP *sends the* BS *to the* HWV) The SWP sends the BS of the purchased application and the $\mathrm{id_{HAP}}$ that bought it. Also in this case, the communication involves sensitive data (BS, $\mathrm{id_{HAP}}$) thus the SWP and the HWV communicate by using a secure channel that ensures confidentiality, data integrity and authentication by means of an agreed symmetric key K_{SH}.

5. (*The* HWV *prepares the* BS *for the client's* HAP) After receiving the data, the HWV ciphers the BS with the HAP key and sends back to SWP the ciphered bitstream $\{\mathrm{BS}\}_{\mathrm{K_{HAP}}}$. In this case, the confidentiality of the secure channel is not needed, only the integrity and authentication. However, the already established secure channel could be used again.

6. (*The* SWP *forwards the encrypted* BS *to the* STORE) the SWP sends back to the STORE the ciphered bitstream $\{\mathrm{BS}\}_{\mathrm{K_{HAP}}}$ via the available secure communication channel.

7. (*The* BS *ready to be downloaded by the* EU *from the* STORE) Finally, the STORE makes available to the EU, e.g., through the user account, both the executable code SW and the ciphered bitstream $\{\mathrm{BS}\}_{\mathrm{K_{HAP}}}$.

The EU is then ready to install the application on its device. After the installation, every time the user starts the application the reconfiguration of the HAP will be triggered. Within the HAP the BS will be decrypted, thanks to the embedded controller that uses the HAP key $\mathrm{K_{HAP}}$, and loaded onto the reconfigurable logic.

It is worth noting that the proposed infrastructure can also be used to deliver updates to the BS, while updates to the executable code can simply be downloaded by the STORE through the usual methods. When the SWP updates the BS, e.g., because of a new version release or bug fixes, a reduced version of the previous protocol can be used:

1. the current version of the application initiates the communication with SWP to check for available updates.

2. If a new update is available, the client sends its $\mathrm{id_{HAP}}$ to the SWP (thus bypassing the STORE) through a secure channel that ensures confidentiality,

data integrity and authentication by means of an agreed symmetric key K_{CS}. The SWP checks if the client corresponds to a valid customer[3].

3. The SWP connects to the HWV via a secure channel and sends to the HWV the BS and the id_{HAP}, encrypted with an agreed symmetric key K_{SH};

4. The HWV ciphers the BS using the HAP key K_{HAP}, and sends back to the SWP the ciphered bitstream $\{\text{BS}\}_{\text{K}_{\text{HAP}}}$. In this case, a secure channel is not needed however, the already established can be used;

5. finally, the SWP sends back to the EU the updated application, which includes the updated ciphered bitstream $\{\text{BS}\}_{\text{K}_{\text{HAP}}}$.

Note that the simple scenario describes a communication protocol for the exchange of the bitstream among several parties that can be easily implemented with network nodes and a suitable End User Device.

5.3.2 Case 2: Advanced scenario

The simple scenario presented in Section 5.3.1 ensures confidentiality, data integrity and authentication among the involved parties. However, it requires a trust relationship between the SWP and the HWV, which might already exists due to legal contracts. Nonetheless, it may be a limitation. Moreover, this protocol does not minimize the disseminated information. For instance, the HWV is aware of each BS application the EU purchases.

In the simple scenario, the EU is able to obtain an application for its device, while the SWP achieves a secure transfer of its IP. However, there is no assurance for the SWP that its IP will not be loaded and executed in compromised hardware, which could ease the porting of attacks. Additionally, the EU privacy could be undermined (and the "need to know" requirement as well), since the other parties may collect information about the software bought by the user from any developer. Instead, in the scenario presented here, either the EU is able to preserve anonymity and the SWP may only sell his application to users that own legitimate hardware. On the one hand, the full scenario drops the assumption that there exists a trust relationship between the SWP and the HWV. On the other hand, it requires that the HAP offers more sophisticated features that are usually available at dedicated secure hardware. Indeed, the protocol implemented in the full scenario relies on the execution of the Direct Anonymous Attestation (DAA) protocol to guarantee all the security requirements in Section 5.2.2.

[3]This check implies that the SWP has stored the list of id_{HAP} of customers sent by the STORE.

The DAA Protocol

The Direct Anonymous Attestation protocol has been designed to allow a verifier to check that a signature has been originated by a legitimate platform by means of a deterministic verification algorithm [35]. Signatures can be used to convince a verifier about the integrity of the platform, that is, signatures allow achieving platform attestation. Signatures made with the DAA protocol are anonymous, that is, they do not leak any information about the signer, unless the signer wants a subset of signatures to be linked. By using the DAA protocol it is also possible to detect the so-called *rogue* platforms, so that the signatures obtained by means of compromised keys can be revoked. The manufacturer of the secure chips providing the DAA protocol can thus maintain a list of the compromised platforms.

The DAA protocol is independent of the public key authentication schemes, thus, different types of keys can be embedded in the secure hardware platform.

Different versions of the DAA protocol have been presented in the last years since the publication of the first version. Some of them were provably secure under assumptions that do not guarantee the claimed security properties in the real world, for some other schemes, there exist known attacks to compromise them [57, 36, 56, 58]. However, recently, two forms of DAA have been presented that have been formally proved against threats models that are not considered weak [40, 41].

The DAA protocol has been standardized by the Trusted Computing Group (TCG) and is available in the Trusted Platform Module (TPM) since the version 1.2[4]. Later, in 2013, the TPM v2.0 has been developed and provided with the most efficient of the published DAA protocol versions. The same version of the DAA protocol has also been inserted in the Intel processors as Enhanced Privacy ID (EPID) algorithm, which has been standardized as ISO/IEC 20008-1:2013[5] and 20009-1:2013[6].

Three roles are involved in the execution of the DAA protocol:

1. the *DAA Issuer* is the entity that manufactures the secure hardware platform. The DAA Issuer knows all the secrets the secure hardware platform stores.

2. The *DAA Signer* is the secure hardware platform that produces the signatures used for attestation purposes. The DAA Signer entity is composed of two parts, the secure hardware platform and a Host, the set of all the software components that interfaces with the secure hardware platform (e.g., the computer system where the secure hardware platform is available, including the OS and all the drivers that allow accessing the hardware). Finally,

[4]However, the version of the DAA protocol implemented in the TPM v1.2 is no longer considered secure.

[5]https://www.iso.org/standard/57018.html

[6]https://www.iso.org/standard/57079.html

3. the *DAA Verifier* is any external party (e.g., a service provider) that is interested in verifying the integrity of the secure hardware platform.

To achieve signature anonymity, the DAA protocol introduces two more features:

- a counter, which is a value used to generate multiple DAA keys from a single secret, and
- the basename: an optional value used to allow verifiers to link multiple DAA signatures signed under the same DAA key. Basenames can be considered pseudonyms and are obtained by means of a secure function computed on the secret of the secure hardware platform.

To abstract from the DAA protocol implementations (and their present and future flaws) and to make our result more general, we assume in this work that the system uses the most secure and efficient DAA protocol version, which will expose the primitives presented below.

- The DAA Setup is the procedure that a secure hardware platform performs to indicate its host and then to the DAA Issuer whether or not it is corrupted.
- The DAA Join is the procedure that a host performs, together with the secure hardware platform, to obtain the DAA Credentials (which can be seen as an anonymous public key certificate) and becomes part of the group of certified or attested secure hardware platforms.
- The DAA Sign/DAA Verify are the two procedures that the secure hardware platform and the host use to convince a verifier that the secure hardware platform is certified and the host has previously joined the group. These signatures are generated from a DAA Credential obtained after the join. This procedure is a form of remote attestation. Nevertheless, the same primitives could also be used to actually sign messages.

The DAA protocol ensures several security properties. They have been formally defined in previous works [40, 41] and are reported in this thesis in an informal way:

- *Completeness*: signatures from a valid basename of a honest platforms are accepted as valid by the verifiers.
- *Correctness of Link*: signatures created with the same basename by the same honest platform are correctly linked.
- *Unforgeability*: no adversary can create a signature that is recognized by the verifiers as a valid signature of a honest platform, regardless of the basename he used.

- *Anonymity*: signatures of the same honest platform that use different basenames (or no basename at all) are not linked by any adversary. In other words, the adversary cannot tell if they are applied by the same honest platforms o by different honest platforms.
- *Non-frameability*: no adversary can create signatures with a given basename that links to a signature created by an honest platform.

Functional assumptions for the advanced scenario

The following functional assumptions have been considered for the full scenario:

- The EU is able to access the store (i.e., has an account on the STORE);
- The EU has a credit card or any another equivalent payment system accepted by the store;
- The STORE has previous agreements with one or more payment systems;
- The HAP owns valid DAA credentials and is able to run the Setup, Join, and Sign DAA methods;
- The SWPs know the HAP manufacturers (i.e., the HWVs) and can access their DAA Issuer services;
- (optionally) the STORE trusts the information obtained by the HWVs through their services (e.g., the information about its compromised HAPs).
- (optionally) the involved entities are able to create secure channels (e.g., through SSL/TLS protocols) satisfying confidentiality, data integrity and authentication, and peer authentication using an agreed symmetric key.

It is worth noting that the functional assumptions just mentioned are quite similar to the ones presented in 5.3.1. However, in this scenario, some entities offer special functionalities and play specific roles with respect to the context of the DAA protocol here employed:

- The HAP plays the role of the secure hardware platform;
- The End User Device plays the role of the host where the secure hardware is attached;
- The HWV plays the role of DAA Issuer (i.e., the TPM manufacturer), thus it maintains the rogue oracle;
- The SWP plays the role of DAA Verifier (i.e., it wants to authenticate the HAP)

- optionally, also the STORE can play the role of the verifier (i.e., if it wants to authenticate the HAPs)

Workflow

In the full scenario, before deploying an application onto a device embedding reconfigurable computing resources, the HAP executes the DAA-Join protocol in order to be recognized as a trustworthy device by the HWV. The target of this phase is to prove to the DAA Issuer that the secure hardware platform is trustworthy (setup) and to obtain valid DAA credentials (join).

Fig. 5.4 shows the workflow for the application deployment[7].

The steps to buy a new application in the full scenario are the following ones (see Fig. 5.4).

1. (*The* EU *buys the application*) the EU browses the STORE and decides to buy an application, composed by a SW and a BS parts, developed by a SWP. The STORE redirects the user on the PG to complete the application purchase. If the payment transaction terminates successfully, the PG informs the STORE. The STORE starts the procedure to send to the EU the requested BS.

2. (The STORE notifies SWP of a new customer) the STORE notifies the SWP that a new customer intend to buy the application (thus he needs the related BS). Since the communication does not involves sensitive data, the STORE and the SWP may also avoid using a secure channel.

3. (*The* EU *sends the* SWP *its data*) During this interaction, the EU sends the information needed to identify its HAP, i.e., the DAA Credentials issued by the HWV (the DAA Issuer) when the DAA-Join was performed. Since the communication includes sensitive data (such as the credit card number for the purchase and the $\mathrm{id}_{\mathrm{HAP}}$), the End User Device and the STORE will use a secure channel that ensures confidentiality, data integrity and authentication.

4. (HAP authentication and optional key exchange) the SWP verifies that the customer owns a genuine HAP by executing the DAA-Verify protocol. To know whether the HAP is rogue, the SWP connects to the HWV services to query the "rogue oracle". If the verification fails, i.e., the DAA Credentials correspond to a HAP that is not genuine or they have been marked as rogue, the protocol is stopped by the SWP. Some management policies will establish

[7]Note that, to avoid making a too complex diagram, we have omitted all the authentication and integrity checks as well as the acknowledgements since they have been already presented in the simple scenario (and because they represent standard mutual authentication schemes that use random numbers [157]).

how to deal with these cases (e.g., ask installation on another platform or simply stop the purchase). If the verification is successful then the payment is finalized. We assume that the interactions between the HAP and the SWP by means of the DAA protocol allow the exchange of a public key used by the SWP to encrypt the data, which will be decrypted only inside the HAP. In case the RSA-DAA protocol is used, based on the strong RSA assumption, the DAA Credential already conveys a certified public key $k_{\text{HAP,pub}}$. In case other public schemes are used, like the ones based on the qSDH and the LSRW assumptions or the ECDAA, we assume the HAP is able to generate an RSA key pair and, via the DAA Sign, it can send the SWP the signed public key $k_{\text{HAP,pub}}$.

5. (The SWP sends the EU the encrypted bitstream) the SWP generates a new symmetric key, the session key K_s, ciphers it using the received $k_{\text{HAP,pub}}$, then it ciphers the bitstream with K_S. Finally, it sends both the $\{K_s\}_{k_{\text{HAP,pub}}}$ and $\{BS\}_{K_s}$ to the STORE[8].

6. (The EU downloads the application from the STORE) finally, the STORE makes available for download to EU from his account both the software SW, and the enveloped data structure including $\{K_s\}_{k_{\text{HAP,pub}}}$ and $\{BS\}_{K_s}$.

The EU is then ready to install the application on its End User Device.

As for the simple scenario, the EU is then ready to install the application on its device. Every time the End User starts the application, the HAP is reconfigured with the soft IP core conveyed with the bitstream. Within the HAP the BS will be decrypted thanks to the embedded controller that deciphers the session key k_s from $\{K_s\}_{K_{\text{HAP,pub}}}$ by using the corresponding key $K_{\text{HAP,pri}}$.

5.4 Hardware architecture and implementation

In this section we present our proof-of-concept for the secure bitstream transfer protocol, detailing the proposed hardware architecture. We consider the End User Device as a heterogeneous mobile device composed of the host platform and the Hardware Acceleration Platform, as shown in Fig. 5.5.

5.4.1 Architecture

The host platform integrates hardware peripherals common for mobile computing devices, such as computational cores (e.g., microprocessors, GPUs, etc.), sensors

[8]The data sent in this interaction can also be wrapped in a single enveloped data structure, e.g., the PKCS#7 enveloped data.

(e.g., gyroscope, accelerometers, proximity sensors, light sensors, etc.), display or screen, memory and storage (e.g., embedded memory, RAM, SD card, etc.) and network adapters for connectivity features. The HAP is embedded into the mobile device. Within the HAP, there is the reconfigurable resource, i.e., the FPGA. The HAP embeds also a microcontroller offering cryptographic functionalities, such as key generations, and to securely load the bitstreams of the applications installed onto the FPGA.

5.4.2 Interconnections

The class of mobile devices is often characterized with components enabling two different types of connections, i.e., wired and wireless. Network adapters and antenna of the End User Device enable external connectivity towards an internet connection, which is necessary for the user to browse application stores and to download the SW and BS of the applications to be installed.

Internal connections depend on the actual hardware architecture of the End User Device. Among the components of the EU device, the data path of the bitstream spans from the storage medium to the FPGA within the HAP. In order to permanently store an application, the host microcontroller is connected to the storage medium. The hardware description of applications is loaded onto the FPGA with a link between the host and the HAP. Eventually, when the FPGA has to be programmed the bitstream is moved to the HAP where the controller decrypts the hardware configuration and sends it to the FPGA.

5.4.3 Implementation details

To simulate our architecture, we separated the host from the HAP of the End User Device. The host device is emulated with a normal laptop/desktop PC connected to the internet, running the client software. The HAP is connected to the host PC with a USB cable. For prototyping the HAP, we employed a particular chip, i.e., SEcube™ [30]. SEcube™ is a heterogeneous system-on-chip, embedding three components interconnected within the same package: a microcontroller, an FPGA and a SmartCard. The microcontroller is a 32-bit low-power ARM Cortex-M4 processor performing the necessary operations to communicate with the host, through USB and SDIO interface. To securely program the FPGA, the microcontroller is connected to the FPGA through a 16-bit wide bus. The reconfigurable hardware is a Lattice MachXO2-7000 low-power FPGA. The SmartCard is Certified Common Criteria CC EAL5+. The host is connected to the storage medium, i.e., a microSD card, which is accessible also from the microcontroller of the HAP. The storage medium is employed to store the bitstream of the applications downloaded. Although the BS on the storage medium can be accesible from outside, it is stored encrypted to avoid confidentiality breaches. The bitstream is decrypted only within

the HAP, which is considered a trusted area. The other involved parties STORE, SWP and HWV are represented with other PCs connected in a network. Each entity executes an application to communicate with the other network nodes and serves the respective clients according to the protocols disccussed in Section 5.3.

5.4.4 Software stack

The software executed on the bare metal of the HAP device is the open source firmware of the SEcube™ SDK. It provides a high-security abstraction layer through API functions, ranging from encryption and decryption utilities to cryptographic keys management for the keys embedded in the HAP, as well as interfaces for secure communication with the host platform. To implement the functionalities required for the DAA protocol of Section 5.3.2, we adopted the MIRACL Crypto SDK [211]. The MIRACL SDK is a C/C++ library providing optimized implementations for security primitives (e.g., elliptic curve cryptography - ECC), tailored for constrained environments, such as embedded systems and mobile devices.

Among the applications running on the host side of the End User Device, the counterpart of the APIs are used to communicate with the HAP. Also, the host device runs the client market application, which is able to connect to the services offered by the Software Application Stores. The server applications of the STORE, SWP and HWV resides in the same PC, working as separate asynchronous processes. The services of these entities have been emulated as Python3 applications. For the client/server architecture and asynchronous communication functionalities we employed the *asyncio* module. The communication among these processes, running on the same physical machine, flows through different port numbers of the communication sockets on the same IP address. Every message sent is encrypted and signed to guarantee integrity, confidentiality and authenticity. When reaching the destination, the signature of the message is checked against malicious alterations. When the download of the bitstream is completed, the application running on the End User Device triggers the mechanism for loading the bitstream onto the FPGA. The HAP firmware retrieves the encrypted data from the storage medium and check its integrity and authenticity. Finally, the bitstream is decrypted and the plaintext configuration is loaded onto the FPGA by the microcontroller equipped onto the HAP.

5.5 Security analysis

From Section 5.2.3, we see two types of attackers to counteract: man-in-the-middle (MITM) and man-at-the-end (MATE) attackers. The implicit assumption in an MITM scenario is that both the endpoints are trusted entities. However, this assumption is no longer valid in the case where also MATE attackers are interested

in obtaining the bitstream. Indeed, MATE attacks are more challenging to prevent. However, all the security relevant operations are performed in the HAP, which we assume is a trusted device that cannot be attacked by our adversaries.

It is worth remarking that the proofs of the security of these solutions hold under the *infeasibility* hypothesis. That is, we assume the use of state-of-the-art secure cryptographic algorithms with proper key lengths. For instance, currently, a 256-bit security is required from symmetric encryption, that is, the best attack should be a brute force attack on a 2^{256} size key space, i.e., using AES256 guarantees an appropriate level of security. Furthermore, at least 80-bit of security is required for the digest algorithms, for instance SHA-256 is currently a valid choice.

Moreover, we recall that when two communicating peers A and B share a symmetric key k:

- (*confidentiality*) only A and B are able decrypt messages encrypted with k;
- (*symmetric integrity and authentication*) if A receives a message and a MAC computed by using k, A deducts that the message and the MAC were generated by B and no one changed the original message, analogous considerations are valid when B receives messages and MACs A.

5.5.1 Simple Scenario

MITM attacks performed by remote adversaries represent a standard security problem in computer networks and there are provably secure solutions to protect against these attacks: the channel protection techniques. In fact, MITM attacks can be neutralized by using strong peer authentication mechanisms to avoid impersonation, symmetric data integrity and authentication techniques to avoid the message forging and alteration, and symmetric data encryption to ensure confidentiality of exchanged data.

The techniques we adopted in the simple scenario protocol ensure data confidentiality by using symmetric encryption algorithms (e.g., AES), and symmetric data integrity and authentication algorithms (i.e., a keyed digest or HMAC using a cryptographic hash function) used in the a "challenge-response (keyed) one-way functions authentication protocol" [157]. These are well-known approaches that are provably secure under the infeasibility assumption.

The first step of the protocol for bitstream IP protection specified in Section 5.3.1 represents a typical e-commerce scenario very widespread nowadays, where a user is assumed to have a credit card, a TLS-enabled browser installed in his environment, and an account on the application store of the platform. The store relies on payment services that should adhere to the Payment Card Industry Data Security Standard (PCI DSS) to work in the financial world [62]. Indeed, the PCI DSS imposes high security requirements for merchants and payment servers that

store, process or transmit payment cardholder data when implementing a robust payment card data security process.

By analysing the protocol, we derive that the $\mathrm{id}_{\mathrm{HAP}}$ is only readable by the EU, the STORE, the SWP, and the HWV. In fact, the HAP identifier is encrypted with the key shared between the client browser and the STORE, then the STORE encrypts it with the key shared with the SWP. Finally, $\mathrm{id}_{\mathrm{HAP}}$ is again encrypted with the key shared with HWV and it is sent to the HWV. MITM attackers cannot read the HAP identifier if strong encryption algorithms are used. Additionally, impersonation attacks are impossible as the sent messages allow symmetric authentication of the party. Furthermore, the data authentication and integrity mechanisms used by the presented protocol (i.e., HMAC or an authenticated encryption algorithm) prevent that modifications to exchanged messages are not detected. Therefore, the only remaining attacks are the DoS attacks, which we have excluded as it is not among our goals (and not easy to prevent in all cases).

Even more important, the BS is read in clear only by the SWP and the HWV. In fact, the bitstream is encrypted with the key shared between the SWP and the HWV. Then, the HWV encrypts the BS by using the secret key K_{HAP}, shared with the HAP of the End User Device. Therefore, no one but the HAP with the identifier exchanged during the protocol is able to load and use it. The received bitstream is stored in an encrypted form until it is moved to the HAP decrypted and loaded onto the FPGA, while the software application is usually downloaded in plaintext, since other software protection techniques are displaced to protect the intellectual property, if needed. Therefore, MITM and MATE attacks that aim at reading the bitstream in intelligible way are avoided.

These considerations prove that only the EUs that bought the software are able to use the corresponding BS. Moreover, even the end users are not able to read the plaintext of the bitstream.

Note that the confidentiality of $\mathrm{id}_{\mathrm{HAP}}$ and the integrity and authentication of all messages could be achieved if, instead of an ad-hoc protocol as in Fig. 5.3, general purpose channel protection mechanisms are used. Even better, these channels usually also provide protection from reply attacks and filtering, by numbering exchanged packets or by storing the last packets. The most widespread ones are the TLS protocol [67], which works at the transport layer of the ISO/OSI stack, or the IPsec protocol [131], which works at network layer, and other application layer methods, usually message protection techniques (e.g., WS-Security). In our case the TLS approach is the preferred one, which better integrates with a web-based scenario that is very frequent in the e-commerce scenarios. Indeed, it does not require additional software or any previous knowledge of the other communicating party but limited modifications of the services or the availability of an ad hoc API.

On the other hand, there is no alternative than the explicit encryption with K_{FPGA} to protect the confidentiality of the bitstream.

5.5.2 Advanced Scenario

The advanced scenario falls in the same common e-commerce use case. We propose an alternative use of DAA protocol in a more general context than the TCG related ones. The only difference is that the user is required to perform a DAA_Setup to initialize the HAP and a DAA_Join before running the protocol to generate and issue new DAA credentials of the FPGA. From the functional point of view, DAA-related operations are only available in experimental settings and have not yet reached a large audience, thus not to current customers. However, they do not pose significant challenges for future EUs, also because they can be mostly automated and their complexity can be properly hidden to end users.

Indeed, after buying the software, the client is redirected towards the SWP and he is asked to perform a DAA_Sign to attest the integrity of its HAP. Only the SWP knows the session key K_s, which is then ciphered with the public key received by HAP. Therefore, K_s will only be available inside the HAP, in the trusted part of the protocol entities. This in turn allows satisfying the "Bitstream confidentiality", "Bitstream integrity and authenticity" and "Legitimate End User" requirements. To complete the integrity verification the SWP will contact the "rogue oracle", that is, the entity knowing the IDs of the compromised HAP. The HWV is the best (and only, to date) candidate to maintain this list.

Note that the generation of an RSA key pair whose public component has to be signed with the DAA Credentials and then sent Software Provider, is currently not implemented in the TPM chips. However, it is not against the TPM and DAA specifications, as DAA_Sign can be both used for remote attestation and data signature purposes. Therefore, from the functional point of view, the operation we require poses integration issues with current TPM chips but it is not a problem for possible future versions of the TPM nor for the secure hardware (FPGA) we are considering for this work, as they are fully reconfigurable.

Therefore, in this scenario the only entity that knows the BS is the SWP, because the HAP can only load and use it internally. Thus, when following this approach the "Need to know" principle of the BS is better ensured as it guarantees that the minimum number of entities know the BS in an intelligible form. Indeed, as the HAP of the End User is the only entity that knows the private component of the public key sent to the SWP, only the HAP of the legitimate buyer can use the BS, thus ensuring the "Legitimate End User".

Additionally, by using the DAA_Sign and DAA_Verify operations and the remote verification using the rogue oracle, the SWP is able to sell its products to buyers whose HAP is not compromised, thus proving the "Legitimate HAP" requirement.

Therefore, we can conclude that the security of the advanced scenario only depends on the correctness and security of the DAA protocol. We assume that the DAA protocol used is correct and secure. Given the level of interest in this protocol

and the effort put by the TCG and the researchers in the field, this is currently a reasonable assumption and it will become even more acceptable also in the near future. Indeed, as discussed in Section 5.3.2, the recent progresses and publications give positive hints on the fact that the DAA development is converging to a form that is both secure and not too demanding from the computational point of view [40, 41].

The idea of using the DAA protocol has major consequences to the impact of the work presented here. Indeed, the use of a well-known protocol guarantees a bigger control on the strength of the secure mechanisms. Moreover, there will be more people researching for flaws, as the impact of attacks becomes very high thus also the impact of publication of such attacks is more likely to reach a higher visibility. Also reactions of the involved parties to potential flaws would have much more support. The TCG is also increasingly working to increase anonimity and improve users's privacy. In our opinion this is a major trend that is worth joining.

We reach at the same time a strong protocol that guarantees a better level of user privacy (i.e., used data and hardware identifiers) and minor exposition of the company IP (soft IP core are only read by developers and used by secure hardware).

Therefore, even if we do not necessarily propose the use of TCG-specified TPM chips, we follow the progresses in this field and re-implement in reconfigurable hardware the same features.

Another good reason to join a mainstream initiative, is a greater level of control on the whole supply chain of the trusted hardware device, being they TPMs of FPGA-based devices. Indeed, since nothing can be done if the HWV inserts backdoors in the produced devices, we have to resort to a trusted supply chain.

Together with simple backdoors to access the content of the HAP to download the IP cores that need to be protected, other attacks against the whole End User Platform can be leveraged by holes in the security of the supposedly trusted device.

As a simple instance of potential attacks, instead of generating a new RSA key pair inside the HAP, the HWV can generate and inject a certain number of RSA key pairs to choose from (or DAA Credentials if RSA-DAA is used). Therefore, the HWV would have access to all bitstreams encrypted with those public keys.

5.5.3 Physical Attacks

Physical attacks represent another type of possible attacks performed directly on the End User Device. In this case, an attacker aiming at finding the symmetric encryption key must be necessarily a MATE owning the device of interest. We distinguish the case of non-invasive physical attacks from invasive physical attacks.

Non-invasive attacks can be partitioned in attacks where the device is purposely *stressed*, i.e., *active*, and attacks where there is no interaction with it, i.e., *passive*. In both cases, the destruction of the device is not necessary. However they generally require a longer time to be accomplished. Brute-force attacks have already

been excluded under the infeasibility hypothesis. Side-channel attacks can still be employed, however they are not always possible and require specialized equipment. The SEcube™ chip adopted for the prototype is secure against the differential power analysis attack as shown in [31]. Timing attacks can be neutralized resorting to constant-time implementation of the bitstream decryption and other security primitives.

Invasive physical attacks destroy the device and require sophisticated and expensive equipment, knowledgeable attackers, and possibly long time to be carried out.

In our scenarios, a physical attack might target the Host platform, the HAP or the storage medium. Any attack to the Host or the storage medium is neutralized with the encryption algorithm, since these parts of the end user platform are considered not trusted. Accessing these peripheral will give to the attacker the possibility to find the ciphertext of the bitstream. But the data must still be deciphered, by reversing the key. The HAP is in fact the critical element, since the bitstream is deciphered within this device through the key stored here. Local adversaries should first bypass any external protection to reach the internal circuitry.

If we suppose that an invasive physical attack is successful, only a single device is compromised. Every device stores a unique serial number and the cryptographic key relative to the specific device. In this way, an attacker wanting to inject vulnerabilities or malicious hardware must repeat the attack on other HAPs. This means that also other devices should be compromised in the same way, leading to higher costs to spread the attack. Also, the effort to exploit the attack is linear with the number of device to compromise.

Nonetheless, if the attack is successful, the target IP stored onto the device is compromised by only compromising a single device. Indeed, when the bitstream is available to the attacker, it can be decrypted and its description might not remain confidential, but could be disclosed to the public. Moreover, this security breach gives the possibility to the attacker to recover also the $\mathrm{id}_{\mathrm{HAP}}$ of the compromised device. By knowing this information, an attacker could also obtain any other bitstream previously bought for that specific device.

Note that if a TPM chip supports the DAA protocol features, this chip is protected by design against side-channel (timing information, power consumption, electromagnetic leaks) and physical attacks. Additionally, as it uses cryptographic operations it must comply the FIPS 140-2 standard.

The TPMs available on the market provide memory curtaining and protected execution to avoid that the MATE reads the stored secrets. Therefore, the only way to read the secrets is physically tampering with the chip. To the best of authors' knowledge, the only successful physical attack is the one presented at the Black Hat conference 2010 [227]. At cost of six months and 200,000$, Tarnovsky tampered the internal circuitry of an Infineon TPM of the SLE 66PE family of contactless interface microcontrollers to get the secrets. The attack required to

dissolve the outer shell with chemicals and remove the layers of mesh wiring to access the chip's bus to read the secrets by tapping the communications channels using small needles. The attack was defined by the author "not easy to duplicate" and the Trusted Computing Group that issued the TPM specifications, a bit more optimistically, "exceedingly difficult to replicate in a real-world environment".

Some other attacks to the TPM are known in literature that are not important for us. The reset attack, is a method to reset the TPM without resetting the entire system. In this way the known-good hashes can be stored in the TPM circumventing the "extend-only" functionality that does not allow to overwrite values in the TPM registers. This attack is mounted against TPM family 1.1 using a vulnerability of the Low Pin Count (LPC) bus that allows to reset all the attached devices. From the family 1.2 this attack is no longer possible.

Another reset attack is presented in [102], however the authors already contributed together with manufactures with a patch to solve the vulnerabilities.

A recent side channel attack on TPM 2.0 devices is presented in [161], where the authors are able to recover 256-bit private keys for ECDSA signatures Intel firmware-based TPM as well as a hardware TPM. The attack exploits timing information leakage and allows key recovery in less than two minutes. Also in these case, the vulnerabilities have been solved.

5.6 Conclusion

This work addresses the protection of soft IP cores to be deployed in the context of mobile heterogeneous systems.

We provide an architecture for a secure transfer of a soft IP core bitstream from a generic software developer to an end user owning a device equipped with reconfigurable logic, considering a common real-world scenario of mobile application market.

The provided protocol details the deployment for two scenarios, considering first a minimum set of security requirements, then an advanced scenario fulfilling tighter security properties.

We guarante that only legitimate users purchasing a legitimate copy of an application are enabled to use it. During the deployment, we are able to maintain the confidentiality of the intellectual property. Also, the integrity of the data transferred is preserved to guarantee that no alteration, whether malicious or not, has been performed. Compliance to these requirements protects from MITM attackers. We considered the threat of MATE attackers as well. Finally we also provided a prototype implementation of the whole architecture, employing a system-on-chip as heterogeneous system to work together with a host device (e.g., a PC, or a mobile device).

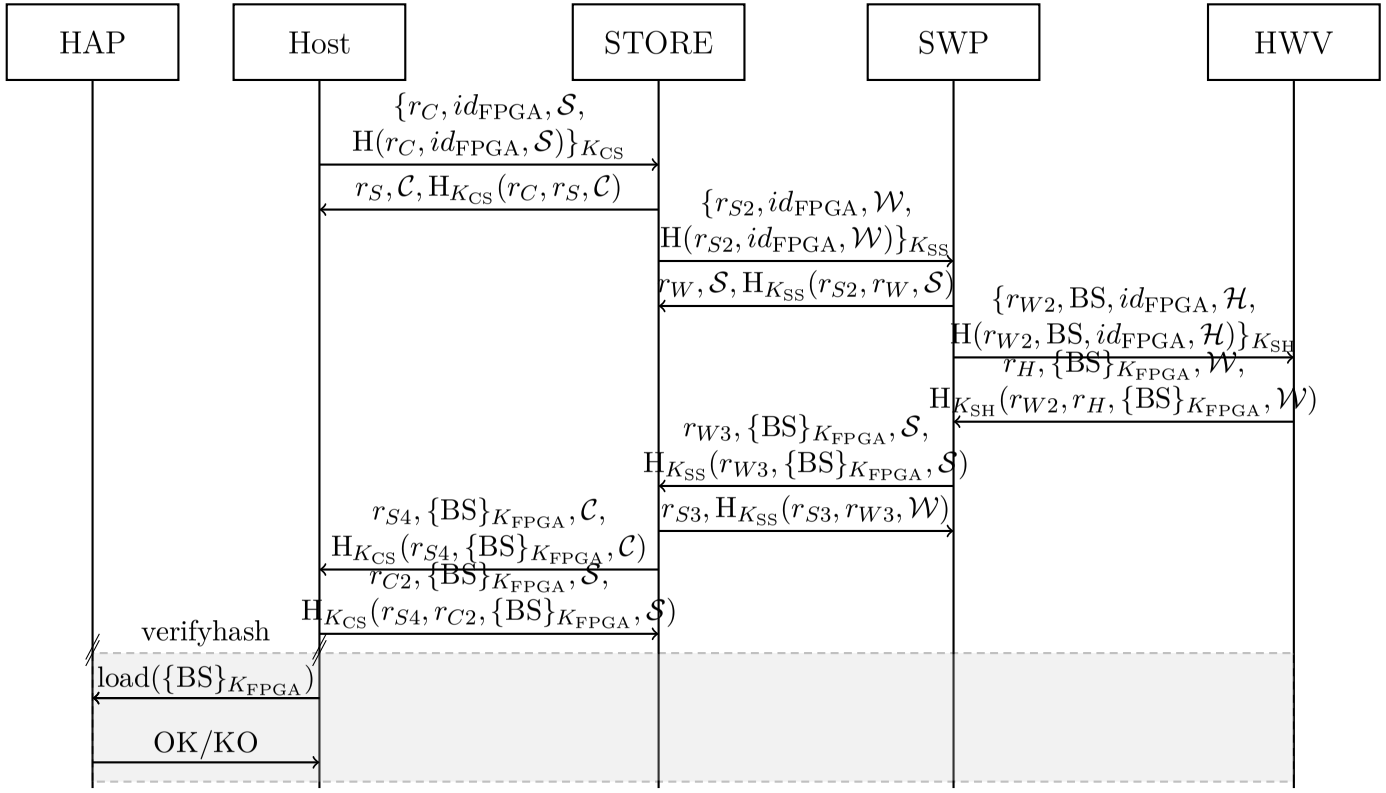

Figure 5.3: The simple case workflow for downloading a new application. The grey area shows messages exchanged locally in the End User Device, i.e., between the host platform and the HAP.

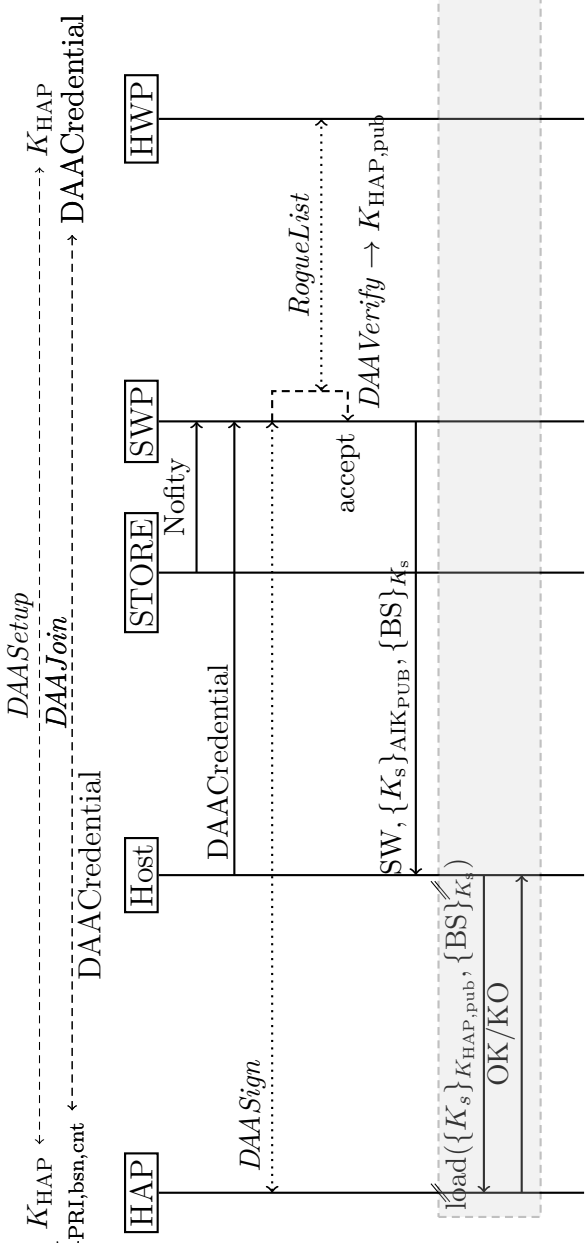

Figure 5.4: The full scenario workflow. The grey area shows messages exchanged locally in the End User Device, i.e., between the host platform and the HAP.

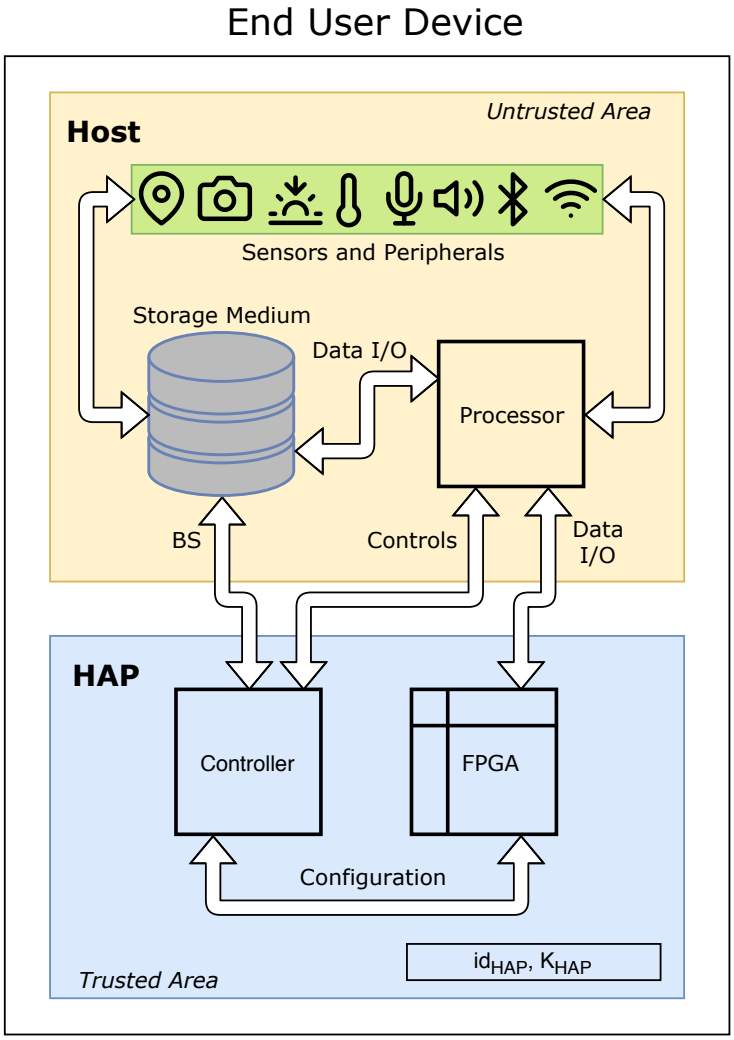

Figure 5.5: The internal architecture of the End User Device.

Chapter 6

HSM Secure Firmware Development and Analysis

This chapter details the development of an open source firmware to create a security platform exposing Hardware Security Modules functionalities. The resulting platform is then analyzed under the security point of view to assess its resilience against a physical side-channel attack.

6.1 Introduction

Devices such as HSM and TPM are of crucial importance in the protection of systems exposed to security threats, such as CPSs and Critical Infrastructures. These devices can secure various sensitive data processing operations and provide robust mechanisms for authentication and cryptographic functions. Further advantages deriving from the integration of these categories of devices into existing systems are described in detail in Sect. 2.6.

However, very often the solutions provided by security modules are proprietary, i.e., the publisher or the vendor of the solution retains the intellectual property rights. This implies that the software or the hardware (or both) is compliant with the closed-source model. In this model, the details of the implementation represent the IP itself. Only generic specifications are released to the public, while the full details of the implementation are kept reserved by the IP owner. On the one hand, usually, the advantages deriving from the adoption of closed solutions have important strenghts, such as improved reliability, warranty and help service for maintenance and repair. On the other hand, in some scenarios, these closed solutions are not a valid option also because of the security concerns. The users, in the closed model, have to be willing to accept the level of security that vendors or licensers provide. In case some vulnerabilities are disclosed, the users are left without any capacity to mitigate the threats. Indeed, they are totally dependent

on creators for a possible upgrade. In case a vulnerability is found in proprietary solutions, several security concerns arise: how it is addressed and how long it takes to be corrected. Moreover, considering the producer retains the IP, it can also decide to not fix it at all. Most importantly, is how the security vulnerabilities are found, especially for what concerns the closed-source software. Indeed, security audit is not always possible on proprietary solutions.

Open solutions can solve these security concerns, leaving capabilities to users to customize and improve the product when needed. The question of security between closed and open solutions is very controversial, because there is no guarantee that open solutions are more secure. However, there is an advantage deriving from openness itself.

Critical Infrastructures do not have to rely exclusively on closed HSM products that cannot be audited and might lead to distrust the functions implemented in some of these products. It is on this principle that an open-source and open-hardware platform can bring additional advantages.

An example of open security device is SEcube™ [1], the hardware platform considered in this work. Its design includes three hardware components and the open source software described in this chapter. The rest of this chapter details the software architecture and its functionalities and provides a security analysis of the whole platform against a side-channel attack. The development of the software aimed at the creation of an open source firmware to be run on the device and a corresponding library[2] to be employed on the host. From the host side, the software produced is an SDK (Software Development Kit) exposing a set of functions that can be employed to exploit the capabilities of the secure platform. To assess the security properties of the SEcube™ platform, a Differential Power Analysis (DPA) attack is performed against the device. DPA is a non-invasive side-channel attack, which considers the power consumption of a circuit. The target of the attack is to identify the key used during the encryption of a data stream. Performing this attack enables to measure the resilience of the device against a physical attack.

6.2 Security Features

The security features deriving from an open platform are diverse. From the hardware side, an open-hardware HSM allows diverse manufacturing by multiple vendors. This means the HSMs can be produced also in different countries and governments. Moreover, for both hardware and software, the openness allows for different designs and development features. Open devices can be replicated anywhere,

[1] https://www.secube.eu

[2] Last updated version of the firmware can be retrieved at https://www.secube.eu/resources/open-sources-sdk/

leading to gain a divese users community. Thus, each subscriber can autonomously audit the security of the device.

SEcube™ is a HSM-like device able to provide basic security features. It is composed of heterogeneous security-oriented hardware coupled with an open-source modular software architecture. All the functional blocks of the platform can be employed to provide additional protection to applications where data privacy and security are critical requirements, e.g., in safety and security critical environments that emphasize data integrity, availability and confidentiality. By being open source, users and developers are able to build, modify, and rewrite the whole system if wanted. Moreover, the open source approach encourages the proliferation of examples and implementations of security architectures. The device is designed and constructed with ease of integration and service-orientation in mind.

SEcube™ is basically an heterogeneous chip that can be embedded on most devices, such as USB token or USB Mass Storage device, or into larger systems. In this way, it can communicate with the host where it is integrated to assist the computations and to provide security mechanisms. The software running on the host communicate with the firmware running on the device. The device provides additional security services, e.g., cryptographic algorithms.

The main functionalities a SEcube™ platform provides are the following:

- Secure key storage: cryptographic keys of custom length can be stored inside the device. This allows for enhanced protection of cryptographic keys. Storing the secret keys within this device prevents their accidental copy and distribution. The key is securely stored in hardware, residing in the flash memory of the heterogeneous chip. The keys never leave the device. Once a key is stored, there is no way to retrieve it.

- Encryption and decryption service: data stream can be encrypted or decrypted employing the device using any of the stored keys. When the key is needed to cryptographic operations, the data is sent to device. This guarantees that the key needed for encryption/decryption does not leave the device. Moreover, this mechanism enables also to unburden the host from the computations deriving from the related security operations by employing the device as a cryptographic co-processor.

- Authentication service: data streams can be authenticated using any of the stored keys. As with data encryption, authentication is also handled on the device. This service employs cryptographic signatures which are depending on a cryptographic key stored within the device. Also in this case, the operations performed on the device lighten the workload of the host.

6.3 Hardware Platform

From the hardware point of view, the SEcube™ device is an heterogeneous platform, providing three different technologies. It is produced by *Blu5 group*[3] in a very small form factor (i.e., BGA 9x9mm). In this way the chip can be embedded in other devices, platforms and existing systems. Considering the form factor, the integration will results in a limited area footprint.

It is an open hardware design composed of three Commercial-Off-The-Shelf (COTS) cores. This allows the users to replicate the design without necessarily purchasing from a specific vendor. In addition, this choice also helps reducing the costs associated with the manufacturing of the device, both for production and prototyping.

SEcube™ is a single System-on-Chip (SoC), composed of three main devices in a single package: (i) a microcontroller, (ii) an FPGA and (iii) a SmartCard.

The first component is the central core of the device. It is a microcontroller unit (MCU), i.e., an ARM Cortex-M4 processor, within a MCU of the STM32F4 family by *STMicroelectronics*[4]. In particular the MCU is an STM32F429. The 32-bit RISC core operates at a frequency of up to 180 MHz and provides 2 MB dual-bank of Flash memory and 256 KB of SRAM. It integrates common devices, such as multiple ADCs and DACs, and communication interfaces, such as USART, I2C and SDIO.

The second component is a flexible and fast Field-Programmable-Gate-Array (FPGA). The FPGA is a MachXO2-7000 by *Lattice Semiconductor*[5]. The FPGA has 6864 LUTs and 47 I/O lines. Another important feature is that the FPGA is an ultra-low power device, which makes it optimal for constrained environments (e.g., heterogeneous mobile devices).

The last component is a SmartCard. The SLJ52G is produced by *Infineon*[6]. The SmartCard is CC EAL 5+ certified, guaranteeing high security standards. It provides 128 KByte EEPROM and offers several encryption algorithms for both symmetric and asymmetric cryptography (i.e., DES, 3DES, AES up to 256-bit and RSA up to 2048-bit, ECC up to 521-bit).

Fig. 6.1 shows the components and how they are connected to each other.

These components make the SEcube™ platform suitable for a wide range of common applications (e.g., telecommunications, IoT, domotics, robotics, etc.) and also for applications where security is a critical issue (e.g., Critical Infrastructures, CPS, ICS, etc.). Moreover, customized security solutions involving cryptographic

[3] https://www.blu5group.com

[4] https://www.st.com

[5] https://www.latticesemi.com

[6] https://www.infineon.com

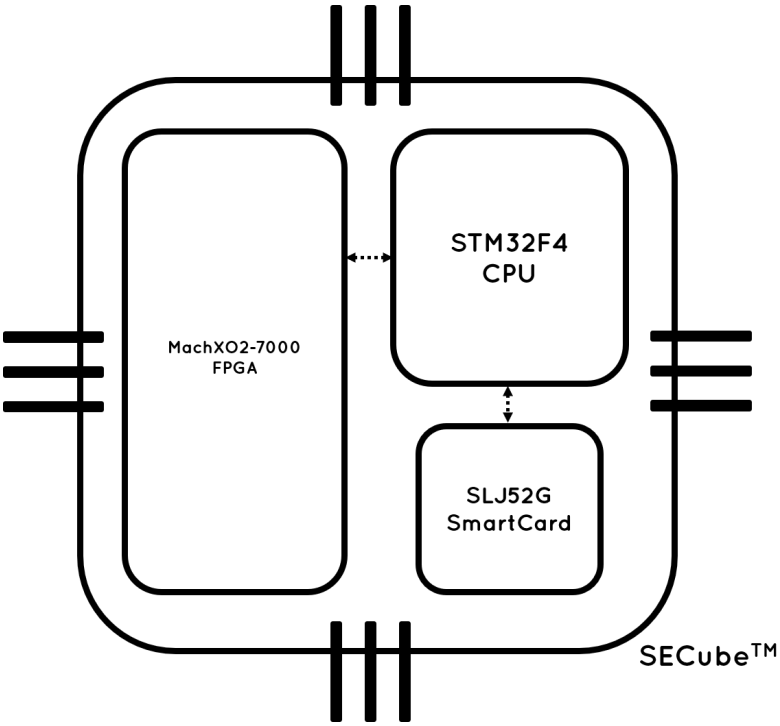

Figure 6.1: High level hardware architecture of the components.

algorithms and additional security services can be implemented resorting to these components.

6.4 Software Architecture

The software part of the platform is composed of two symmetrical parts, depending on where physically the code runs:

- *Device-side software:* This code runs on the device, i.e., the chip. It includes software drivers to interface with the underlying hardware peripherals, cryptographic primitives and the device-side functions implementation.
- *Host-side software:* This code is a library of functions that run on the host (e.g., a normal PC). It is logically composed of different abstraction layers, exposing the functionalities of the device to the host.

The device code is the firmware running on the MCU of the platform. It commands the various peripherals and communication interfaces. Thus, it is able to communicate with the other devices, either internal to the chip (i.e., FPGA and SmartCard) or external (e.g., host, microSD card).

The firmware integrates the drivers for the peripherals. The majority of the peripheral drivers are automatically generated with the assistance of STM32CubeMX[7] software. This tool allows a simple and immediate configuration of STM32 microcontrollers. According to the configuration, it is able to generate the initialization source code for the device. Moreover, the firmware implements the core functions used for cryptographic operations against a data stream. The encryption keys also reside on the device, and the core library provides the function for managing the keys as well.

To use the functionalities of the hardware, the users (or developers) of the host need to interface with the device. The host-side software realizes the communication. Moreover, it provides a library of functions organized in a layered architecture. The applications that want to exploit the functionalities existing on the security device can resort to this library.

The layered architecture of the software exposes the device-side API to the host. The SDK provides two layers, i.e., Layer 0 (L0) and Layer 1 (L1). These two layers provide the basic mechanisms to interface with the device. The library is implemented by relying on levels of hierarchical abstraction. L0 provides function primitives to the higher levels. In addition to L0 and L1, increasing levels, such as Level (L2), employ the functions of lower levels. Fig. 6.2 reports a graphical representation of the software architecture.

Each level targets a specific set of functionalities to be offered to the host:

- L0: provides basic functionalities to interact with the device.
- L1: provides basic security functionalities, such as encryption services.
- L2: provides services to ease the integration of the security features of the SEcube™ hardware in the development of new applications.

the SEcube™ DevKit[8] has been employed for the development of the firmware and the host library. The development board embeds the SEcube™ chip and several other components, such as LEDs and physical connectors. For example, it provides USB plugs for connecting to an host peripheral. In addition, there is a microSD card reader connected to the SDIO interface of the MCU within the SEcube™ chip. Finally, it exposes connections for hardware debugging through JTAG interface.

.

[7] https://www.st.com/en/development-tools/stm32cubemx.html

[8] https://www.secube.eu/products/secubetm-development-kit/

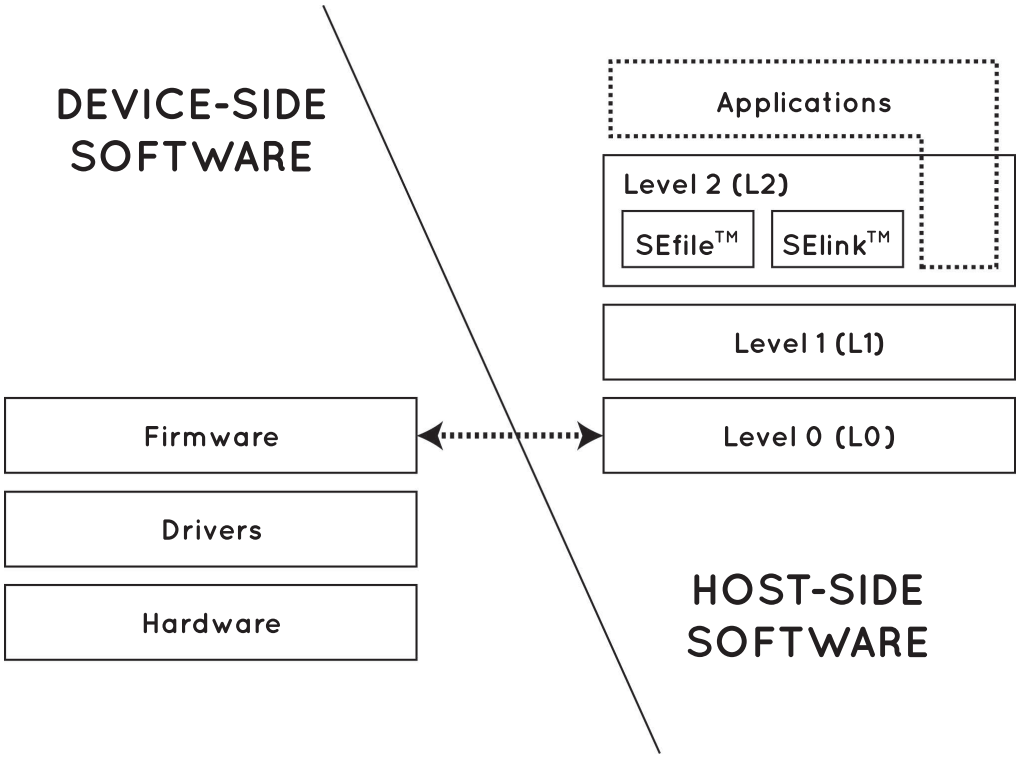

Figure 6.2: High level software architecture.

6.4.1 Firmware - Core

Low-level communication protocol

The firmware of the SEcube™ device is configured in such a way the a SEcube™ device is seen by the host as a USB CDC Mass storage device, like a normal USB flash drive. The mass storage is represented by a microSD card inserted in the respective reader of the development board. Thus, the host operations on the SD are forwarded to the SDIO interface of the MCU. However, some of the block write operations are filtered and decoded as special commands sent from the host to access the SEcube™ cryptographic functions. Similarly, some block read operations do not actually read data contained on the microSD, but read the response generated by the command instead. The communications with the host are implemented via a special file, which is read and written from both ends. The communication between device and host is based on a request-response protocol, where the host sends a command (*request*) to be executed by the device, by writing a special file. The device interprets the command and returns its response (*response*) by writing this special file. The host initiates the communication by writing the special file, formatted in a particular way, on the SD. The host writes in this special file a command to be executed on the device. Then, the same file is polled for the

response, by consecutive reads, until the response is finally ready or until a timeout event occurs. Fig. 6.3 depicts the communication.

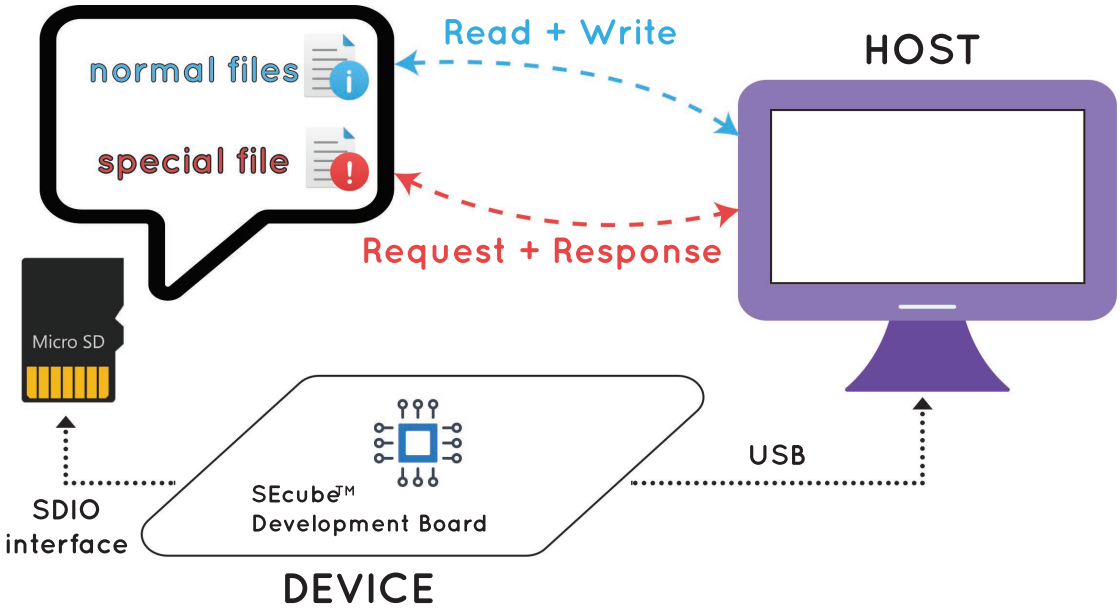

Figure 6.3: Basic Request-Response communication protocol.

In order to communicate with the device, the host has to find it among all those plugged. This process takes the name of discovery. The host creates (or tries to create) the special file on all the detected devices. During the discovery, the special file created is populated replicating a 32-byte sequence. The total size of the file is 8 KB (8192 bytes). The file is logically divided in 16 blocks of 512 bytes each.

Only the SEcube™ devices answer to the discovery process. Upon recognizing the repeated pattern, the device maps each block of the special file in its internal memory. Each block is employed by the device to read and to answer the commands of the host. To answer the discovery process, the device alters the last block. The written fields are device-specific (such as the serial number, vendor identifier, firmware version, etc.) and customizable. For the host, the 16th block is used in read-only access for passive operations, such as the discovery process. This operation only needs a block read. No blocks are written by the host, thus avoiding conflict with other pending operations on the device or concurent access by other applications. The communication mechanism is based on a request-response protocol. The format of the data exchanged is fixed. There are different levels of commands sent/received. Each level is encapsulated in the payload of the lower level.

User authentication and Accounts

In order to exploit the services of the security module, the owner must be authenticated on the device. All commands sent to the device before a successful log in are rejected and the device remains locked. The same applies in case the device has not been initialized correctly. The firmware supports two different accounts type, i.e., User and Administrator. These accounts corresponds to the following access levels: (1) read access and (2) write access. Each access level limits the operations performed on the configuration records, that are generic user settings stored within the device memory. A successful log in enables the use of the services provided by the device, according to the access level. Once logged a communication session respecting the privilege of the access level is established between the SEcube™ and the host.

The log in operation employs a challenge-based scheme for authentication with a passphrase. The passphrase for the different accounts is first set, during the device initialization. During the authentication a session key is exchanged, thus protecting the session confidentiality.

The data related to the communication session is stored in the memory of the MCU. To avoid dynamic allocation, memory regions to store session data are statically allocated. In case no contiguous memory is available, but still enough free memory is available, the session data is fragmented and the new session is accommodated. The session is freed when the user perform the logout operation.

Flash memory management

The information to be secretly preserved by the HSM are stored within the MCU Flash memory of SEcube™ . The information can be written during the factory initialization (i.e., manufacturer, device ID, passphrases, etc.) and during the life of the device, i.e., cryptographic keys. To store the information two 128 KB sectors of flash memory are used. However, in flash memories it is not possible to modify the value of a single bit. Instead a full sector erase is required, incurring in a time overhead and in a reduced lifespan of the flash memory. To tackle this problem, in the event that a piece of information is deleted the corresponding region is invalidated. Between the two sectors, only one is the active sector being used. The other one is used as backup to store data when erasing the active sector. This swap occurs also in the case it is necessary to store additional data, but there is not enough free space available due to invalid blocks. If the request can be satisfied, a swap of the two sectors is triggered. The valid data is copied from the active sector to the backup sector, the backup sector becomes the active one and finally the request is accommodated.

Each 128 KiB flash sector is divided into 64-byte blocks. The sector contains also additional metadata information. The sector is organized according to Table 6.1.

Table 6.1: Flash sector layout

Offset [B]	Size [B]	Name	Description
0	32	*marker*	active/backup marker
32	2016	*index*	index table type for data blocks
2048	129024	*data*	effective data blocks (64 bytes each)

The marker sequence is used to discriminate the active sector from the backup sector. The index represents each 64-byte block with a byte where the type of the corresponding data block is stored. Custom predefined values for the type allow to discriminate among a free block, an invalid block or a block that contains data spanning over multiple blocks. In this last case, the data size is written in the first 2 bytes of the data. The rest of the memory contains the actual data blocks. The rationale behind this scheme is that if data inside a record is corrupted, next records will be still accessible. This also holds true for any corrupted bytes in the index.

Cryptographic Keys

Keys are basically data structures stored within the flash memory of the MCU within SEcube™ . The fields composing the key are the following:

- id: key unique identifier;
- data: the actual key data;
- name: a string to describe the key;
- validity: deadline of the key.

This structure is stored in the flash memory as a variable size data. However, the maximum lengths for key data and key name are 2048 bytes and 32 bytes, respectively. A timestamp is associated to each key (i.e., validity field): after this time has expired, the key is no longer valid and cannot be used anymore.

FPGA

The FPGA within the SEcube™ chip can be employed to host one or more IP cores. Each core can implement a set of functions realizing enhanced versions of algorithms to boost computational performance or provide additional functionalities enabling new capabilities for the HSM. The FPGA is connected to the MCU within the platform. Thus, each core instantiated on the FPGA should provide its own software driver to be embedded in the firmware. The target of the driver is

to manage the communication mechanism with its complementary part residing in the reconfigurable fabric. Fig. 6.4 shows the bus lines interconnecting FPGA and MCU.

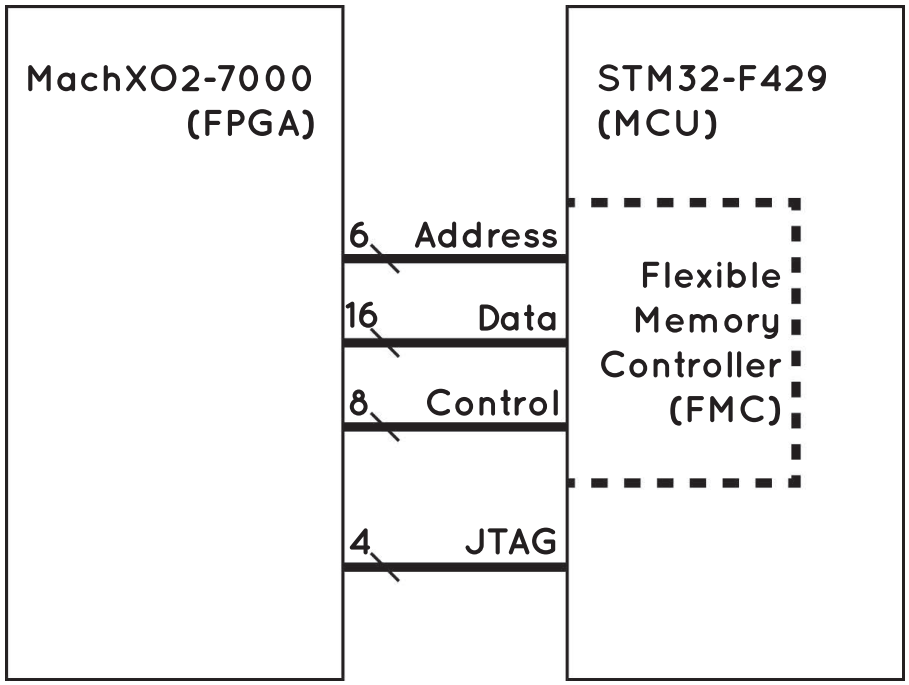

Figure 6.4: Communication bus between MCU and FPGA.

The set of interconnections is used for exchanging data, control, and status signals.

The default configuration of the MCU enables to interface with the FPGA as an external memory device. This functionality exploits the Flexible Memory Controller (FMC) available on the MCU. The values and the bit transitions on the communication lines are directly managed by the FMC. Employing this feature, simplifies the transmission of data between the MCU and the core(s) residing into the FPGA.

The JTAG interface is employed for programming the FPGA. The vendor of the FPGA provides the mechanism to program the reconfigurable devices using a microcontroller in embedded systems, such as SEcube™ . The vendor provides device independent software source files written in ANSI C language, which can be compiled and executed with the SEcube™ firmware and the files containing the FPGA design.

In the SEcube™ firmware, the bitstream configuration mechanism is modified to securely load the bitstream. In this way, the final FPGA design can be encrypted before the deployment in-the-field. The bitstream is then decrypted only within the heterogeneous chip with a key residing in the MCU.

6.4.2 Host library

The Host library is structured as set of API functions that enable a host device to interface with the SEcube™ HSM. In this way, the communication and the operations between the host and the SEcube™ device are simplified. These functions are structured in levels of increasing abstraction.

Level 0 - L0

The Level 0 (L0) exposes the basic communication functions with the device. It allows to:

- detect and interface with the SEcube™ peripherals connected,
- low-level request-response communication protocol with the device.

With these functions, a developer can open and close the communication with the device without interfacing directly with the device and without manually crafting the messages to be sent.

Moreover, this level enables functionalities for the factory initialization of a SEcube™ device. This operation allows to set the device-specific information (e.g., serial number) and to set a passphrase for the users of the device, or add default cryptographic keys. Ideally, they need to be performed only once in the life-cycle of the device. The information of the factory initialization cannot be reversed without resetting the internal flash memory of the MCU.

For diagnostic purposes, also the *echo* functionality is available at this level. This function sends a piece of data and wait for the reply from the device. The reply contains an exact copy of the data sent. This function does not require the authentication of an user of the device.

Level 1 - L1

The Level 1 (L1) of the library exposes the basic security functionalities of the device to the host. These functionalities can be employed to implement secure applications running on the host.

The functionalities provided at this level allow:

- authentication of an user on the device,
- key management operations, and
- encryption/decryption of data streams.

All the messages exchanged are encapsulated in the L0 communication protocol. To use any of the functions belonging to this level, the user needs to be authenticated on the device. The authentication lets an user, either admin or normal user,

log into the device. It requires a passphrase, which can also be changed resorting to functions of this level.

Key management functions affects cryptographic keys stored in the device. The functions are employed to add, modify or delete a key. Moreover the host can retrieve a list of all the keys stored.

This level exposes also the primitives of the cryptographic algorithms available on the device. The functions encrypt or decrypt the data with a key stored within the SEcube™ device. However, the functions realizing this feature exploit the device as a cryptographic accelerator. The data to be encrypted (or decrypted) is sent to the device and it is retrieved decrypted (or encrypted). Moreover, the data can also provide signature for authenticity purposes.

The supported encryption algorithm is AES-256. Table 6.2 shows the modes of operations for this cipher supported in this firmware.

Table 6.2: List of supported mode of operation for AES

Cipher	Modes of Operation
AES	Electronic codebook (ECB) Output feedback (OFB) Cipher block chaining (CBC) Cipher feedback (CFB) Counter (CTR)

The digest algorithm SHA-256 is employed as message authentication code (MAC) to provide authentication. It can be used to produce data digests or to produce a keyed signature, i.e., a Keyed-hash message authentication code (HMAC). Moreover, authentication can be combined with encryption to produce a single payload. In this case, the *Encrypt-then-MAC* scheme is used: the plaintext is encrypted and the MAC is computed on the resulting ciphertext. Table 6.3 summarizes the cryptographic primitives supported in this firmware.

Table 6.3: List of cryptographic primitives supported

Cryptographic Primitive	Type	Description
AES256	Encryption (only)	Encryption with AES cipher
SHA256	Digest (only)	Hash digest with SHA-2
HMACSHA256	Signature (only)	Keyed hash with SHA-2
AES-HMACSHA256	Encryption and Signature	Data encryption with AES and signature with HMACSHA256

Level 2 - L2

The Level 2 (L2) library enables services for secure storage (SEfile™) and secure transmission of data across unprotected networks (SElink™) [246]. It combines the functionalities provided by the layers below to realize services that can be transparently used by other applications.

SElink™

SElink™ is a software that uses the SEcube™ platform to secure the network traffic. This application is basically a filter that intercept and encrypt network data streams. The connections can be created by any existing applications on top of an OS, regardless of the application-level protocol. This service allows to add a security layer to the network stack, without modifying the applications. It adds the security features of the SEcube™ platform in a transparent way to the user and to the developer of applications.

SElink™ is an application composed of two parts:

- *Client:* is the software installed on the host side that initiates the communication. This service intercepts the outgoing connections and redirects the network stream to the encryption layer. Then, the encrypted network stream is routed to the destination.
- *Server:* is the software installed on the host that accepts the connection. The service is symmetrical to the client, i.e., it receives the encrypted network stream, decrypts it and routes it to the destination service.

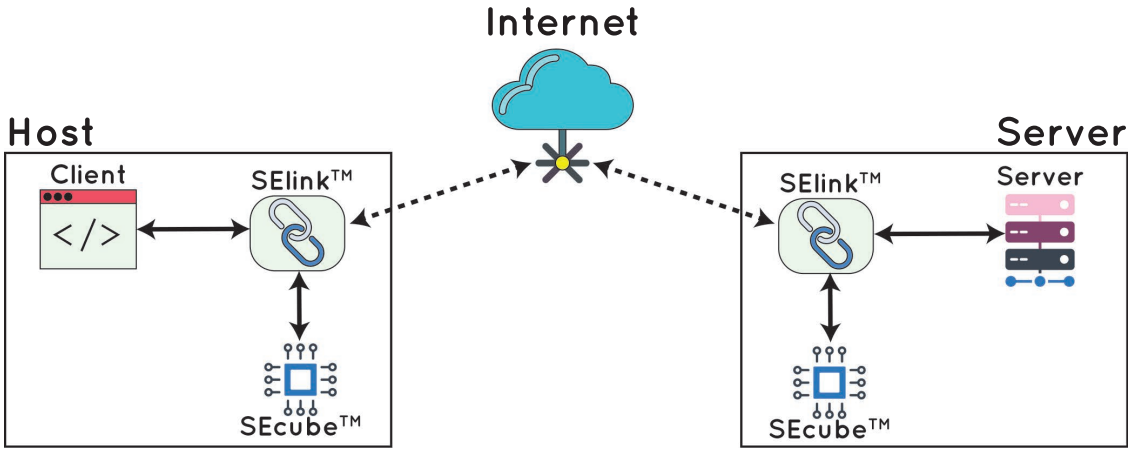

Figure 6.5: Secure encryption layer architecture.

Fig. 6.5 shows the architecture of SElink™. The functionalities provided by SElink™ can be used also as a standalone service and are built on top of the L1 library.

SEfile™

SEfile™ is a host-side library that employs the SEcube™ platform to secure data. This library represents an abstraction layer for secure file manipulation. Instead of dealing with data stream, as lower levels, this library deals directly with files.

A developer can exploit the functions of this level to perform the basic operations (i.e., open and close, read and write) on regular files in an encrypted manner. These operations deal with files on a normal file system. The file is splitted in small sectors and is encrypted or decrypted resorting to the SEcube™ platform. From the user point of view the whole process of encryption/decryption is totally transparent and is implemented in the functions provided.

This library relies on L1 functionalities. Thus, every cryptographic operation required for file manipulation is entrusted to the security module.

6.5 Security Analysis

The remainder of this section is an extended and revised version of the publications [31].

6.5.1 DPA

Differential Power Analysis is a side-channel attack which, considers the power consumption of a circuit. It is a non-invasive attack that requires physical access to the security device. More precisely, to the device supply line. This technique can be employed to identify the key used for the encryption in an encryption algorithm. It can be helpful to identify just a part of the key when this technique is adopted together with a brute-force attack. This can be simplified avoiding unnecessary combinations already assessed with DPA leading to a faster attack.

The basic idea of the DPA is to find a correlation among different sets of data to be processed with a cryptographic algorithm through a statistical analysis of the power consumption values [136]. It exploits the fact that, different data lead to a different power consumption when the operations performed are the same.

First, several sets of power consumption measurements have to be collected avoiding as much as possible noise that will impact negatively the analysis leading to incorrect results. Second, the DPA takes place: all collected power measurements are partitioned in two different sets according to a selection function and with an assumption on the value of a bit belonging to the key. The difference of the average of power traces in each group is computed. This is repeated for every bit of the key.

The selection function determines how to group power traces. It depends on the power consumption model. Several models are possible, in this thesis are considered

only the following:

- Hamming Distance - one set contains the traces that are considered to generate the transition of the bit value 0 to 1 and 1 to 0. The other set contains the remaining traces;
- Hamming Weight - one set contains the traces that are considered to generate the transition of the bit value 0 to 1 and 1 to 1. The other set contains the remaining traces;
- Rising - one set contains the traces that are considered to generate the transition of the bit value 0 to 1. The other set contains the remaining traces;

By construction, one of the two sets contains a power consumption component not present in the other set. The difference of the averages approaches zero for an increasing number of traces when there is no correlation, while it presents a peak in the opposite case. It has to be noted that even in presence of noise affecting the measurements, given enough traces it can be possible to detect small correlations [136]. The choice of the selection function will change the components of the two sets, and as a consequence also the averages change. In general, this leads to different results in terms of correct bits.

6.5.2 Setup

The security device we use in our experiments is the SEcube™ platform. It is an open-source security platform which provides both hardware schematic and software source code. Even though the SEcube™ platform is provided with its open-source firmware and the implementation of its cryptographic algorithms is known, in order to better analyze the strength of its security, we decided to employ a black-box approach based on the sampling of the power consumption. Also, we want to compare the efficacy of the DPA attack with a normal device, to verify the security improvement brought by SEcube™ comparing it with a similar device. To make the experiment meaningful, we picked a ST Microelectronics Nucleo board[220] equipped with the same microprocessor[221] of SEcube™ . All these devices are powered through the USB interface, where we act to precisely measure the power consumption.

The commercial USB token USEcube™ does not allow direct access to the internal circuitry. This already partially prevents the usage of probes to capture the power supply signal, unless invasive attacks are employed. For our experiments we adopted the Development Board version. With this device, having direct access to the V_{dd} pins, we could sample the power consumption very easily and with low noise. Nevertheless, to emulate a real attack, for both Nucleo board and SEcube™ board, we decided to use a custom connector to sample the power signal from the

power supply line between the platform and the host PC. This special custom device is tailored for a generic USB device, so it can be employed also for other types of devices. To build it, we used an electronic prototyping board (i.e., a stripboard), where an USB male socket and a USB female connector have been soldered and linked together. The connection between the two ports is voluntarily left exposed, allowing the probes to be attached easily without introducing distortion. Moreover, a BNC connector is inserted in parallel between the supply line (+5V) and ground line (GND) (see Figure 6.6). In this way, this device can be connected directly to an oscilloscope avoiding the noise introduced by probes ohmic contacts, thus leading to more accurate measurements of the input voltage. Also, it allows us to perform the attack without physically modifying the security device, hence maintaining a non-invasive approach. To perform measurements, we employed an oscilloscope. The

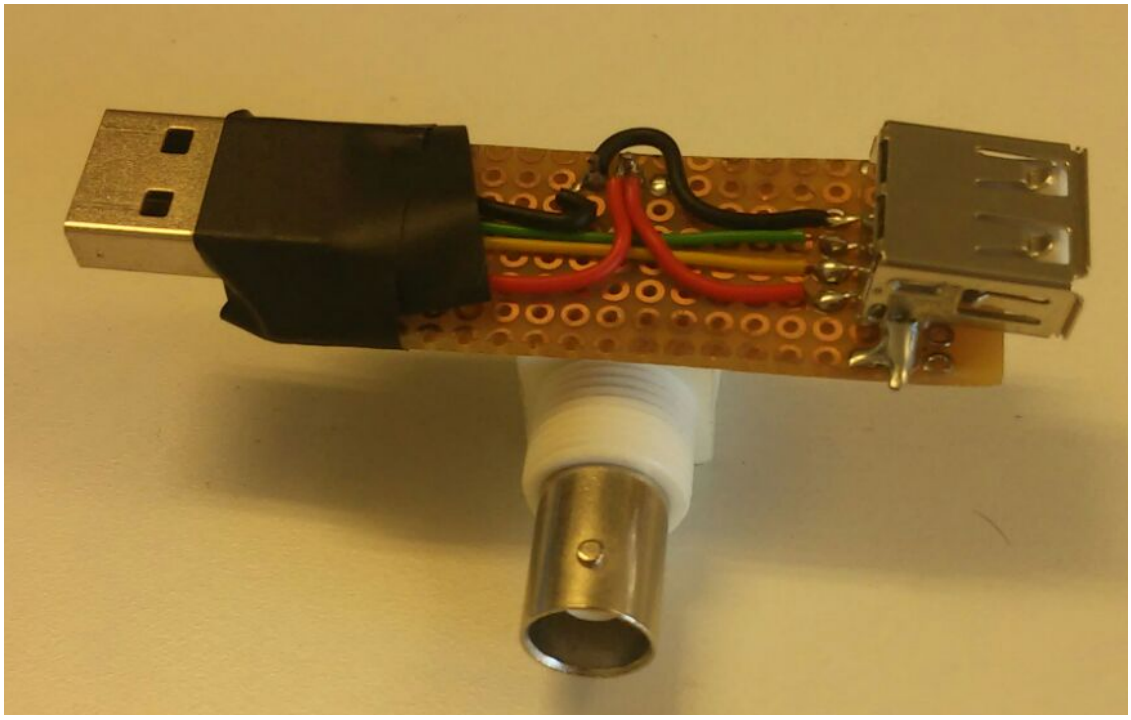

Figure 6.6: Custom handmade USB-USB/BNC connector. Figure taken from [31]. ©2017 IEEE.

used device is a Tektronix TDS5052B [229] with 500 MHz bandwidth, 5 GS/s real-time sample rate and 2 acquisition channels. It embeds a General Purpose Interface Bus (GPIB) controller, which we use to manage the measurement process. In order to collect and store the power traces in an automated way, we setup a LabView module, which interfaces with and command the oscilloscope through the GPIB interface. Also the data is collected through the same physical interface.

Another requirement to perform a DPA attack is to know what algorithm is

involved. Being the SEcube™ firmware open-source we can see that the available encryption algorithm is the AES-256 implemented using a software routine.

To verify the security solely of the hardware platform we created a custom firmware. This firmware includes low-level drivers for the microprocessor but it only executes the encryption algorithm instead of the whole SEcube™ open-source firmware. By employing this new firmware, we can better isolate the encryption process, which leads us to more accurate measurements without spurious transitions due to other operations performed in the original firmware that are not directly related to the encryption. The implemented AES algorithm is coded in C language, like the rest of the firmware. The implementation uses the simplest operation mode, i.e., Electronic Codebook (ECB). Both the used plaintext and the encryption key have size equal to 128 bits and are hard-coded in the firmware. The encryption process is repeated multiple times: it ciphers the plaintext with the same key, but in every round the last byte of the plaintext is changed, in order to have data dependence. The voltage drop at the power supply pins is measured during this process. To simplify the data trace acquisition procedure we added a trigger signal to highlight the beginning of the encryption process.

Finally, the acquired tracks are parsed and edited to be compliant with the software for the DPA analysis realized by the authors of [165].

Figure 6.7 shows the whole workflow just described.

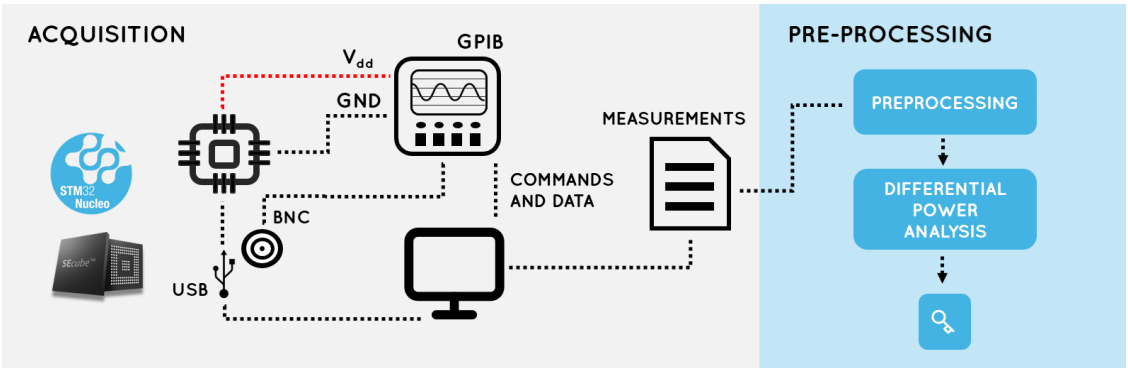

Figure 6.7: Overview of the workflow adopted.

6.5.3 Results

In order to test the security improvement introduced by SEcube™ platform, we adopted a comparative method. We take as a reference design the Nucleo board which embeds the same microprocessor of the SEcube™ . The AES encryption algorithm is executed on both development boards against the same input data. The power traces are sampled with the same experimental setup. The attacks are performed separately and the results are compared. The difference in terms of effort

(number of power traces required, number of guessed bits, ...) gives a quantitative estimation of the security enhancement of the SEcube™ platform with respect to the Nucleo board.

The test-bench algorithm is a software implementation of the AES algorithm. We adopt the same key and the same plaintext (both of 128 bits) proposed in the NIST standard specifications presented in FIPS PUB 197 [85]. However in our experiments, we limit the attack to the last byte of the key. The key and the plaintext used are reported respectively in Table 6.4 and Table 6.5.

Table 6.4: Cipher Key of 128 bit from [85] - Case A. Table taken from [31]. ©2017 IEEE.

0x2b	0x7e	0x15	0x16	0x28	0xae	0xd2	0xa6
0xab	0xf7	0x15	0x88	0x09	0xcf	0x4f	0x3c

Table 6.5: Plaintext of 128 bit from [85] - Case A. Table taken from [31]. ©2017 IEEE.

0x32	0x43	0xf6	0xa8	0x88	0x5a	0x30	0x8d
0x31	0x31	0x98	0xa2	0xe0	0x37	0x07	0x00 .. 0xFF

The encryption process consists into cypher the block of data with the specified key. This is repeated 256 times and in every cycle we vary the plaintext altering the last byte for every possible configuration, from 0x00 to 0xFF. We control and force the changes into the plaintext in order to generate on purpose data-correlation among the sets of measurements. The encryption process is then repeated a modest number of times, in order to have statistical relevance.

The measurements acquired with the oscilloscope amount to 2.713 traces for Nucleo Board and to 15.872 for SEcube™ . The traces are collected during the encryption routine. We decided to acquire more measurements on the SEcube™ platform since it should be more secure. In total, the encryption sessions are 16 for the Nucleo board, against 62 for SEcube™ . Every power consumption trace contains 10.000 samples.

The robustness of the SEcube™ platform against side-channel analysis has been verified through a real DPA attack experience. Both configurations, Nucleo and SEcube™ Development Board, while encrypting test data are analyzed in order to collect the power consumption traces. To relax the time required for running the data acquisition, the attack was limited to a specific byte, thus only 256 traces in every encryption session.

After the acquisition phase, the traces are preprocessed, parsed and fed to the DPA tool. The analysis has been carried out considering variations on the following parameters:

- Number of samples of each trace: we considered 25%, 30%, 50% and 100% with respect to the full number of samples.
- Type of the Algorithm: we considered the whole AES encryption algorithm and the single S-Box (*SboxAES*).
- Selection Function: Hamming weight (*hweight*), Hamming distance (*hdistance*), Rising

Table 6.6 and Table 6.7 present the results from the analysis. Each columns represent the number of samples in every trace. On the rows there are the selection functions considered. Moreover, the results are divided considering whole AES encryption and the single S-Box. The results presented in the various configurations of the parameters state the number of incorrect bits in the last byte of the encryption key. The average of incorrect bits is rounded up and is computed considering the previous configurations. An higher number of bits indicates an unsuccessful attack, which can be interpreted as an enhancement in security.

The reference configuration was attacked first. The results are reported in Table 6.6. It can be noticed that attacking only the Sbox of the AES the results are

Table 6.6: Nucleo results - wrong bits in the key. Table taken from [31]. ©2017 IEEE.

Nucleo

	No. of Samples	2500	3333	5000	10000
SboxAES	*Hamming Distance*	4	2	2	3
	Hamming Weight	5	3	5	6
	Rising	3	4	4	3
	Average	4	3	4	4
AES	*Hamming Distance*	4	2	5	3
	Hamming Weight	5	3	5	6
	Rising	6	4	6	5
	Average	5	3	5	5

slightly better. However, in most cases, just half of last byte of the key is correct. Considering the whole AES, the number of wrong bits increases.

The same attack is performed on the SEcube™ configuration. The results of the analysis are shown in Table 6.7. Also considering the SEcube™ platform the results are similar. The same considerations applies also.

6.5.4 Discussion

From the results of our analysis it can be seen that on average both platforms provide the same level of security. We can state that after analysing several power

Table 6.7: SEcube™ results - wrong bits in the key. Table taken from [31]. ©2017 IEEE.

SEcube					
	No. of Samples	2500	3333	5000	10000
SboxAES	*Hamming Distance*	4	3	4	4
	Hamming Weight	6	4	3	4
	Rising	1	2	1	5
	Average	4	3	3	4
AES	*Hamming Distance*	5	5	5	5
	Hamming Weight	6	4	3	4
	Rising	5	4	3	2
	Average	5	4	4	4

consumption traces we were able to discover roughly only half byte of the whole key. This result might lead to improvements when DPA is combined with a brute-force attack. Although the results are similar, in this work several simplifications are considered. It must be pointed out that the number of traces acquired for the SEcube™ platform is much higher than the one acquired for the Nucleo board. Further investigation is required to achieve more significant results.

6.5.5 Acknowloedgments

The development of the firmware presented in this paper was a team work. I would like to thank Nicola Ferri, Giulio Scalia and Carlo Alberto Cristofanini that collaborated with me at this part of my Ph.D. project.

Chapter 7

Conclusions

This thesis has presented the main activity realized during my Ph.D. studies. The work of research carried out during the three years Ph.D. program aimed at the development and the assessment of new methodologies to increase the cyber security of Critical Infrastructures.

New trends and new technologies have revolutionized the previous generations of architectures, computing systems and networks commonly employed in Critical Infrastructures. Several technologies, such as those specified in Chapter 2, are integrated in these infrastructures making them heterogeneous at different levels.

However, the security domain remains of crucial importance for Critical Infrastructures. As discussed in Chapter 3, new technological trends have generated cyberattacks to become increasingly more complex and more dangerous. Attacks targeting Critical Infrastructures can be considered as acts of cyberterrorism. In these cases, the impairment or the breakdown of a Critical Infrastructure might lead to national matters of public health safety.

Counteracting these attacks is challenging due to the complexity and heterogeneity of Critical Infrastructures. Nevertheless, the integration of security mechanisms is necessary. However, the inclusion of security measures could lead to possible concerns in the safety of a system, which represents another important aspects of Critical Infrastructures. The main contribution of the research activity has been the interplay between these two critical aspects. As discussed in Chapter 4, the mitigation of microarchitectural side-channel attacks requires careful security strategies to avoid additional issues on the safety domain. It derives that those aspects must be considered together because of their mutual interactions.

The usage of heterogeneous technologies in Critical Infrastructures provides additional computational capabilities and enables system flexibility through reconfigurable platforms. On the other hand, new security threats emerge from such technological integration. In particular, the environments employing FPGA-based systems are susceptible to security attacks concerning the bitstream. The bitstream

can be intercepted and altered either during the deployment phase by external attackers or during normal operations by internal local adversaries. These problems were addressed, in Chapter 5, resorting to a secure transmission protocol able to guarantee appropriate security measures for the bitstream against malicious attackers. This technique employs functionalities similar to those in present in hardware security modules and can be employed also for the deployment of reconfigurable design in mobile heterogeneous platforms.

Eventually, the development of a security-oriented open-source firmware for a security platform was detailed in Chapter 6. The resulting platform can be employed as an alternative hardware security module. By being open-source additional security primitives and customized algorithms can be integrated, in contrast to the limitation of closed source solutions. In order to assess the robustness of the security platform, this was subjected to a physical and non-invasive side-channel attack. In detail, the Differential Power Analysis side-channel attack was performed against the device targeting the encryption keys stored within the hardware chip.

Author's Publication List

[1] M. Bollo et al. "Side-channel analysis of SEcube(tm); platform". In: *2017 IEEE East-West Design Test Symposium (EWDTS)*. Sept. 2017, pp. 1–5. DOI: 10.1109/EWDTS.2017.8110067.

[2] A. Carelli, A. Vallero, and S. Di Carlo. "Performance Monitor Counters: Interplay Between Safety and Security in Complex Cyber-Physical Systems". In: *IEEE Transactions on Device and Materials Reliability* 19.1 (2019), pp. 73–83.

[3] A. Carelli, A. Vallero, and S. Di Carlo. "Shielding Performance Monitor Counters: a double edge weapon for safety and security". In: *24th IEEE International Symposium on On-Line Testing and Robust System Design*. 2018.

[4] A. Carelli et al. "Securing bitstream integrity, confidentiality and authenticity in reconfigurable mobile heterogeneous systems". In: *2018 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR)*. 2018, pp. 1–6.

[5] A. Carelli et al. "Securing Soft IP Cores in FPGA based Reconfigurable Mobile Heterogeneous Systems". In: *arXiv preprint arXiv:1912.00696* (2019).

[6] A. Carelli et al. "Towards Model Driven Design of Crypto Primitives and Processes". In: *Proceedings of the International Conference on Security and Management (SAM)*. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp). 2016, p. 152.

[7] A. Vallero, A. Carelli, and S. Di Carlo. "Trading-off reliability and performance in FPGA-based reconfigurable heterogeneous systems". In: *2018 13th International Conference on Design & Technology of Integrated Systems In Nanoscale Era (DTIS)*. IEEE. 2018, pp. 1–6.

[8] A. Vallero et al. "Bayesian models for early cross-layer reliability analysis and design space exploration". In: *2019 IEEE 25th International Symposium on On-Line Testing and Robust System Design (IOLTS)*. IEEE. 2019, pp. 143–146.

[9] A. Varriale et al. "SEcube (TM): Data at rest and data in motion protection". In: *Proceedings of the International Conference on Security and Management (SAM)*. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp). 2016, p. 138.

List of Acronyms

Acronyms / Abbreviations

ADC	Analog-to-Digital Converter
AES	Advanced Encryption Standard
ASIC	Application Specific Integrated Circuit
BBRAM	Battery backed RAM
BHB	Branch History Buffer
BPU	Branch Prediction Unit
BS	Bitstream file
CC	Common Criteria for Information Technology Security Evaluation
CCC	Clock Cycle Counter
CDF	Cumulative Distribution Function
CI	Critical Infrastructure
CIA	Confidentiality, Integrity and Availability
CII	Critical Information Infrastructure
CLB	Configurable Logic Blocks
COTS	Commercial-Off-The-Shelf
CPS	Cyber-Physical System
CPU	Central Processing Unit
CRC	Cyclic Redundancy Checks
DAA	Direct Anonymous Attestation

DAC	Digital-to-Analog Converter
DCM	Data Cache-Miss Counter
DDoS	Distributed Denial of Service
DHS	Department of Homeland Security
DoS	Denial of Service
DPA	Differential Power Analysis
DPR	Dynamic Partial Reconfiguration
DSA	Digital Signature Algorithm
DSP	Digital Signal Processor
EAL	Evaluation Assurance Level
ECI	European Critical Infrastructure
ESS	Emergency Services Sector
EU	End User
EUD	End User Device
FPGA	Field-Programmable Gate Array
GPGPU	General-purpose computing on Graphics Processing Units
GPU	Graphics Processing Unit
HAP	Hardware Acceleration Platform
HMAC	Hash-based Message Authentication Code
HPC	High Performance Computing
HSM	Hardware Security Module
HWV	Hardware Vendor
ICS	Industrial Control System
ICT	Information and Communications Technologies
IIoT	Industrial Internet-of-Things

IoT	Internet-of-Things
IP	Intellectual Property
LLC	Last-Level Cache
LUT	Look-Up Table
MAC	Message Authentication Code
MATE	Man-at-the-End
MCU	Microcontroller Unit
MITM	Man-in-the-Middle
MMU	Memory Management Unit
OS	Operating System
OTP	One-Time-Programmable
PCI DSS	Payment Card Industry Data Security Standard
PG	Payment Gateway
PLC	Programmable Logic Controller
PMC	Performance Monitor Counter
PPD	Presidential Policy Directive
RSA	Rivest–Shamir–Adleman
SCA	Side-Channel Attack
SCADA	Supervisory Control and Data Acquisition
SDIO	Secure Digital Input Output
SDK	Software Development Kit
SHA	Secure Hash Algorithms
SIMD	Single Instruction stream, Multiple Data stream
SoC	Systems on Chips
SPA	Simple Power Analysis

SSA	Sector Specific Agency
SSL	Secure Sockets Layer
SWP	Software Provider
TCG	Trusted Computing Group
TDW	Task Distribution Window
TLB	Translation-Lookaside Buffer
TLS	Transport Layer Security
TPM	Trusted Platform Module
TTM	Time-To-Market
USART	Universal Synchronous/Asynchronous Receiver/Transmitter
VM	Virtual Machine
WSN	Wireless Sensors Networks

Bibliography

[1] M. F. B. Abbas et al. “Hardware performance counters based runtime anomaly detection using SVM”. In: *2017 TRON Symposium (TRONSHOW)*. Dec. 2017, pp. 1–9. DOI: 10.23919/TRONSHOW.2017.8275073.

[2] D. G. Abraham et al. “Transaction security system”. In: *IBM Syst. J.* 30.2 (Mar. 1991), pp. 206–229. ISSN: 0018-8670. DOI: 10.1147/sj.302.0206. URL: http://dx.doi.org/10.1147/sj.302.0206.

[3] Onur Acıiçmez, Billy Bob Brumley, and Philipp Grabher. “New results on instruction cache attacks”. In: *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer. 2010, pp. 110–124.

[4] Onur Acıiçmez and Çetin Kaya Koç. “Microarchitectural Attacks and Countermeasures”. In: *Cryptographic Engineering*. Ed. by Çetin Kaya Koç. Boston, MA: Springer US, 2009, pp. 475–504. ISBN: 978-0-387-71817-0. DOI: 10.1007/978-0-387-71817-0_18. URL: https://doi.org/10.1007/978-0-387-71817-0_18.

[5] Onur Acıiçmez and Çetin Kaya Koç. “Trace-Driven Cache Attacks on AES (Short Paper)”. In: *Information and Communications Security*. Ed. by Peng Ning, Sihan Qing, and Ninghui Li. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 112–121. ISBN: 978-3-540-49497-3.

[6] Actel. *Actel ProASIC3 Handbook*.

[7] A. Adetomi, G. Enemali, and T. Arslan. “Towards an efficient intellectual property protection in dynamically reconfigurable FPGAs”. In: *2017 Seventh International Conference on Emerging Security Technologies (EST)*. 2017, pp. 150–156.

[8] A. Ahmad et al. “3D Haar wavelet transform with dynamic partial reconfiguration for 3D medical image compression”. In: *Proc. IEEE Biomedical Circuits and Systems Conf. BioCAS 2009*. 2009, pp. 137–140. DOI: 10.1109/BIOCAS.2009.5372064.

[9] Manaar Alam et al. “Performance Counters to Rescue: A Machine Learning based safeguard against Micro-architectural Side-Channel-Attacks”. In: ().

[10] Peggy Albright. *Become a Mobile Apps Innovator: Picking an OS and learning to monetize are key.* URL: http://www.computer.org/portal/web/buildyourcareer/HS26 (visited on 2011).

[11] Cristina Alcaraz and Sherali Zeadally. "Critical infrastructure protection: Requirements and challenges for the 21st century". In: *International journal of critical infrastructure protection* 8 (2015), pp. 53–66.

[12] Altera. *Design Security in Stratix III Devices.* URL: www.altera.com/literature/wp/wp-01010.pdf (visited on 2012).

[13] Altera Corp. *AN357: Error detection using CRC in Altera FPGA devices.* URL: http://www.altera.com/literature/an/an357.pdf (visited on 2012).

[14] Grangeat Amélie et al. "The challenge of critical infrastructure dependency modelling and simulation for emergency management and decision making by the civil security authorities". In: *International Conference on Critical Information Infrastructures Security.* Springer. 2015, pp. 255–258.

[15] D. Amelino, M. Barbareschi, and A. Cilardo. "An IP Core Remote Anonymous Activation Protocol". In: *IEEE Transactions on Emerging Topics in Computing* 6.2 (Apr. 2018), pp. 258–268. ISSN: 2168-6750. DOI: 10.1109/TETC.2016.2624026.

[16] Uchenna D Ani et al. "A review of critical infrastructure protection approaches: improving security through responsiveness to the dynamic modelling landscape". In: (2019).

[17] M. N. Anwar, M. Nazir, and K. Mustafa. "Security threats taxonomy: Smart-home perspective". In: *2017 3rd International Conference on Advances in Computing,Communication Automation (ICACCA) (Fall).* Sept. 2017, pp. 1–4. DOI: 10.1109/ICACCAF.2017.8344666.

[18] S. Avramenko et al. "An Hybrid Architecture for consolidating mixed criticality applications on multicore systems". In: *2015 IEEE 21st International On-Line Testing Symposium (IOLTS).* July 2015, pp. 26–29. DOI: 10.1109/IOLTS.2015.7229823.

[19] Benoît Badrignans et al. "SARFUM: Security Architecture for Remote FPGA Update and Monitoring". In: *ACM Trans. Reconfigurable Technol. Syst.* 3.2 (May 2010), 8:1–8:29. ISSN: 1936-7406. DOI: 10.1145/1754386.1754389. URL: http://doi.acm.org/10.1145/1754386.1754389.

[20] S. S. Banu and S. Sonoli. "A Safe Driving Embedded System Integrated with Can Protocol". In: *2017 IEEE 7th International Advance Computing Conference (IACC).* Jan. 2017, pp. 408–411. DOI: 10.1109/IACC.2017.0091.

[21] Alessandro Barenghi and Gerardo Pelosi. "Side-Channel Security of Superscalar CPUs: Evaluating the Impact of Micro-Architectural Features". In: *Proceedings of the 55th Annual Design Automation Conference*. DAC '18. San Francisco, California: Association for Computing Machinery, 2018. ISBN: 9781450357005. DOI: 10.1145/3195970.3196112. URL: https://doi.org/10.1145/3195970.3196112.

[22] Elaine Barker and Allen Roginsky. "Recommendation for cryptographic key generation". In: *NIST Special Publication* 800 (2012), p. 133.

[23] C. Basile, S. Di Carlo, and A. Scionti. "FPGA-Based Remote-Code Integrity Verification of Programs in Distributed Embedded Systems". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42.2 (Mar. 2012), pp. 187–200. ISSN: 1558-2442. DOI: 10.1109/TSMCC.2011.2106493.

[24] Daniel J Bernstein. "Cache-timing attacks on AES". In: (2005).

[25] Sarani Bhattacharya and Debdeep Mukhopadhyay. "Who watches the watchmen?: Utilizing Performance Monitors for Compromising keys of RSA on Intel Platforms". In: *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer. 2015, pp. 248–266.

[26] Meenakshi Bhrugubanda. "A Review on Applications of Cyber Physical Systems". In: *International Journal of Innovative Science, Engineering & Technology* 2.6 (June 2105), pp. 728–730.

[27] Eli Biham and Adi Shamir. "Differential fault analysis of secret key cryptosystems". In: *Annual international cryptology conference*. Springer. 1997, pp. 513–525.

[28] Matt Bishop. *Computer Security*. Addison-Wesley Professional, 2018.

[29] Arnab Kumar Biswas, Dipak Ghosal, and Shishir Nagaraja. "A survey of timing channels and countermeasures". In: *ACM Computing Surveys (CSUR)* 50.1 (2017), pp. 1–39.

[30] Blu5 View. *SEcube(tm) Data Sheet Rev. 7 DUI 15082DS*. URL: https://www.secube.eu/site/assets/files/1145/secube_datasheet_-_r7.pdf (visited on 2019).

[31] M. Bollo et al. "Side-channel analysis of SEcube(tm); platform". In: *2017 IEEE East-West Design Test Symposium (EWDTS)*. Sept. 2017, pp. 1–5. DOI: 10.1109/EWDTS.2017.8110067.

[32] Joseph Bonneau. *aes_cache*. https://github.com/jcb82/aes_cache. 2014.

[33] Joseph Bonneau and Ilya Mironov. "Cache-Collision Timing Attacks against AES". In: *Proceedings of the 8th International Conference on Cryptographic Hardware and Embedded Systems*. CHES'06. Yokohama, Japan: Springer-Verlag, 2006, pp. 201–215. ISBN: 3540465596. DOI: 10.1007/11894063_16. URL: https://doi.org/10.1007/11894063_16.

[34] L. Bossuet, G. Gogniat, and W. Burleson. "Dynamically configurable security for SRAM FPGA bitstreams". In: *Proc. 18th Int. Parallel and Distributed Processing Symp.* 2004. DOI: 10.1109/IPDPS.2004.1303128.

[35] Ernest F. Brickell, Jan Camenisch, and Liqun Chen. "Direct anonymous attestation." In: *ACM Conference on Computer and Communications Security*. Ed. by Vijayalakshmi Atluri, Birgit Pfitzmann, and Patrick Drew McDaniel. ACM, 2004, pp. 132–145. ISBN: 1-58113-961-6.

[36] Ernie Brickell, Liqun Chen, and Jiangtao Li. "Simplified Security Notions of Direct Anonymous Attestation and a Concrete Scheme from Pairings". In: *Int. J. Inf. Secur.* 8.5 (Sept. 2009), pp. 315–330. ISSN: 1615-5262. DOI: 10.1007/s10207-009-0076-3. URL: http://dx.doi.org/10.1007/s10207-009-0076-3.

[37] Samira Briongos et al. "Cache misses and the recovery of the full AES 256 key". In: *Applied Sciences* 9.5 (2019), p. 944.

[38] Eric J Byres, Dan Hoffman, and Nate Kube. "On shaky ground–a study of security vulnerabilities in control protocols". In: *Proc. 5th American Nuclear Society Int. Mtg. on Nuclear Plant Instrumentation, Controls, and HMI Technology* (2006).

[39] Ning Cai, Jidong Wang, and Xinghuo Yu. "SCADA system security: Complexity, history and new developments". In: *2008 6th IEEE International Conference on Industrial Informatics*. IEEE. 2008, pp. 569–574.

[40] Jan Camenisch, Manu Drijvers, and Anja Lehmann. "Anonymous Attestation Using the Strong Diffie Hellman Assumption Revisited". In: *Trust and Trustworthy Computing - 9th International Conference, TRUST 2016, Vienna, Austria, August 29-30, 2016, Proceedings*. 2016, pp. 1–20. DOI: 10.1007/978-3-319-45572-3_1. URL: https://doi.org/10.1007/978-3-319-45572-3%5C_1.

[41] Jan Camenisch, Manu Drijvers, and Anja Lehmann. "Universally Composable Direct Anonymous Attestation". In: *Proceedings, Part II, of the 19th IACR International Conference on Public-Key Cryptography — PKC 2016 - Volume 9615*. New York, NY, USA: Springer-Verlag New York, Inc., 2016, pp. 234–264. ISBN: 978-3-662-49386-1. DOI: 10.1007/978-3-662-49387-8_10. URL: http://dx.doi.org/10.1007/978-3-662-49387-8_10.

[42] Claudio Canella et al. “A systematic evaluation of transient execution attacks and defenses”. In: *28th USENIX Security Symposium (USENIX Security 19)*. 2019, pp. 249–266.

[43] A. A. Cardenas, S. Amin, and S. Sastry. “Secure Control: Towards Survivable Cyber-Physical Systems”. In: *2008 The 28th International Conference on Distributed Computing Systems Workshops*. June 2008, pp. 495–500. DOI: 10.1109/ICDCS.Workshops.2008.40.

[44] Alvaro A Cárdenas, Saurabh Amin, and Shankar Sastry. “Research Challenges for the Security of Control Systems.” In: *HotSec*. 2008.

[45] A. Carelli, A. Vallero, and S. Di Carlo. “Performance Monitor Counters: Interplay Between Safety and Security in Complex Cyber-Physical Systems”. In: *IEEE Transactions on Device and Materials Reliability* 19.1 (2019), pp. 73–83.

[46] A. Carelli, A. Vallero, and S. Di Carlo. “Shielding Performance Monitor Counters: a double edge weapon for safety and security”. In: *24th IEEE International Symposium on On-Line Testing and Robust System Design*. 2018.

[47] A. Carelli et al. “Securing bitstream integrity, confidentiality and authenticity in reconfigurable mobile heterogeneous systems”. In: *2018 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR)*. 2018, pp. 1–6.

[48] A. Carelli et al. “Securing Soft IP Cores in FPGA based Reconfigurable Mobile Heterogeneous Systems”. In: *arXiv preprint arXiv:1912.00696* (2019).

[49] A. Carelli et al. “Towards Model Driven Design of Crypto Primitives and Processes”. In: *Proceedings of the International Conference on Security and Management (SAM)*. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp). 2016, p. 152.

[50] S. D. Carlo, P. Prinetto, and A. Scionti. “A FPGA-Based Reconfigurable Software Architecture for Highly Dependable Systems”. In: *2009 Asian Test Symposium*. 2009, pp. 125–130.

[51] Kate Carruthers. “Internet of things and beyond: Cyber-physical systems”. In: *IEEE Internet of Things Newsletter* 10 (2014).

[52] Defense Use Case. “Analysis of the cyber attack on the Ukrainian power grid”. In: *Electricity Information Sharing and Analysis Center (E-ISAC)* 388 (2016).

[53] Jen-Chieh Chang, Ting-Hsuan Chien, and Rong-Guey Chang. "A Cyber Physical System with GPU for CNC Applications". In: *Algorithms and Architectures for Parallel Processing*. Ed. by Guojun Wang et al. Cham: Springer International Publishing, 2015, pp. 203–212. ISBN: 978-3-319-27137-8.

[54] A. Chatzidimitriou et al. "RT Level vs. Microarchitecture-Level Reliability Assessment: Case Study on ARM(R) Cortex(R)-A9 CPU". In: *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. June 2017, pp. 117–120. DOI: 10.1109/DSN-W.2017.16.

[55] Hong Chen. "Applications of cyber-physical system: a literature review". In: *Journal of Industrial Integration and Management* 2.03 (2017), p. 1750012.

[56] Liqun Chen. "A DAA Scheme Requiring Less TPM Resources". In: *IACR Cryptology ePrint Archive* 2010 (2010), p. 8. URL: http://eprint.iacr.org/2010/008.

[57] Liqun Chen, Paul Morrissey, and Nigel P. Smart. "Pairings in Trusted Computing". In: *Pairing-Based Cryptography – Pairing 2008*. Ed. by Steven D. Galbraith and Kenneth G. Paterson. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 1–17. ISBN: 978-3-540-85538-5.

[58] Liqun Chen, Dan Page, and Nigel P. Smart. "On the Design and Implementation of an Efficient DAA Scheme". In: *Smart Card Research and Advanced Application*. Ed. by Dieter Gollmann, Jean-Louis Lanet, and Julien Iguchi-Cartigny. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 223–237. ISBN: 978-3-642-12510-2.

[59] Y. Chen, V. Mooney, and S. Grijalva. "A Survey of Attack Models for Cyber-Physical Security Assessment in Electricity Grid". In: *2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC)*. Oct. 2019, pp. 242–243. DOI: 10.1109/VLSI-SoC.2019.8920326.

[60] C. S. Collberg and C. Thomborson. "Watermarking, tamper-proofing, and obfuscation - tools for software protection". In: *IEEE Transactions on Software Engineering* 28.8 (Aug. 2002), pp. 735–746. ISSN: 0098-5589. DOI: 10.1109/TSE.2002.1027797.

[61] Mauro Conti et al. *Detecting Covert Cryptomining using HPC*. 2019. arXiv: 1909.00268 [cs.CR].

[62] PCI Council. *PCI DSS Requirements and Security Assessment Procedures, version 3.2.(2016)*. Tech. rep. 2016.

[63] Abhijit Davare et al. "METROII: A design environment for cyber-physical systems". In: *ACM Transactions on Embedded Computing Systems (TECS)* 12.1s (2013), pp. 1–31.

[64] S. Di Carlo et al. "A novel methodology to increase fault tolerance in autonomous FPGA-based systems". In: *2014 IEEE 20th International On-Line Testing Symposium (IOLTS)*. 2014, pp. 87–92.

[65] S. Di Carlo et al. "Dependable Dynamic Partial Reconfiguration with minimal area time overheads on Xilinx FPGAS". In: *2013 23rd International Conference on Field programmable Logic and Applications*. Sept. 2013, pp. 1–4. DOI: 10.1109/FPL.2013.6645549.

[66] S. Di Carlo et al. "Microprocessor fault-tolerance via on-the-fly partial reconfiguration". In: *2010 15th IEEE European Test Symposium*. 2010, pp. 201–206.

[67] T. Dierks and E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.2*. RFC 5246. IETF, 2008.

[68] Derui Ding et al. "A survey on security control and attack detection for industrial cyber-physical systems". In: *Neurocomputing* 275 (2018), pp. 1674–1683.

[69] Jianguo Ding et al. "CPS-based Threat Modeling for Critical Infrastructure Protection". In: *SIGMETRICS Perform. Eval. Rev.* 45.2 (Oct. 2017), pp. 129–132. ISSN: 0163-5999. DOI: 10.1145/3152042.3152080.

[70] Council Directive. "114/EC of 8 December 2008 on the identification and designation of European critical infrastructures and the assessment of the need to improve their protection". In: *Official Journal of the European Union L* 345.75 (2008), pp. 23–12.

[71] Artyom Dogtiev. *App Stores List*. https://www.businessofapps.com/guide/app-stores-list. 2019 (accessed July 8, 2019).

[72] M. Domański et al. "Fast depth estimation on mobile platforms and FPGA devices". In: *2015 3DTV-Conference: The True Vision - Capture, Transmission and Display of 3D Video (3DTV-CON)*. July 2015, pp. 1–4. DOI: 10.1109/3DTV.2015.7169365.

[73] M. Domínguez-Morales et al. "Frames-to-AER efficiency study based on CPUs performance counters". In: *Proceedings of the 2010 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS '10)*. July 2010, pp. 141–148.

[74] Charalampos Doukas et al. "Enabling data protection through PKI encryption in IoT m-Health devices". In: *Bioinformatics & Bioengineering (BIBE), 2012 IEEE 12th International Conference on*. IEEE. 2012, pp. 25–29.

[75] Saar Drimer. "Authenticated of FPGA bitstreams: why and how". In: *In Applied Reconfigurable Computing* 4419 (2007), pp. 73–84.

[76] R. Druyer et al. "A survey on security features in modern FPGAs". In: *2015 10th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*. June 2015, pp. 1–8. DOI: 10.1109/ReCoSoC.2015.7238102.

[77] M. E. Dunham et al. "High Efficiency Space-Based Software Radio Architectures: A Minimum Size, Weight, and Power TeraOps Processor". In: *Proc. Int. Conf. Reconfigurable Computing and FPGAs ReConFig '09*. 2009, pp. 326–331. DOI: 10.1109/ReConFig.2009.42.

[78] Samuel East et al. "A Taxonomy of Attacks on the DNP3 Protocol". In: *International Conference on Critical Infrastructure Protection*. Springer. 2009, pp. 67–81.

[79] Stefano Esposito et al. "A Novel Method for Online Detection of Faults Affecting Execution-Time in Multicore-Based Systems". In: *ACM Trans. Embed. Comput. Syst.* 16.4 (May 2017), 94:1–94:19. ISSN: 1539-9087. DOI: 10.1145/3063313.

[80] Directorate-General for Research European Commission and Innovation. *Cybersecurity in the European Digital Single Market*. Mar. 2017. URL: https://ec.europa.eu/research/index.cfm?pg=newsalert&year=2017&na=na-240317&pk_campaign=policy_newsletter.

[81] P. Falcarin et al. "Exploiting code mobility for dynamic binary obfuscation". In: *2011 World Congress on Internet Security (WorldCIS-2011)*. Feb. 2011, pp. 114–120.

[82] James P Farwell and Rafal Rohozinski. "Stuxnet and the future of cyber war". In: *Survival* 53.1 (2011), pp. 23–40.

[83] S. Ke-fei. "Application of FPGA in Aerospace Remote Sensing Systems". In: *OME Information* (2010).

[84] PUB FIPS. "140-2". In: *Security Requirements for Cryptographic Modules* 25 (2001).

[85] *FIPS PUB 197, Advanced Encryption Standard (AES)*. U.S.Department of Commerce/National Institute of Standards and Technology. 2001.

[86] Apostolos P Fournaris, Lidia Pocero Fraile, and Odysseas Koufopavlou. "Exploiting hardware vulnerabilities to attack embedded system devices: a survey of potent microarchitectural attacks". In: *Electronics* 6.3 (2017), p. 52.

[87] A. Gaware and S. B. Dhonde. "A survey on security attacks in wireless sensor networks". In: *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*. Mar. 2016, pp. 536–539.

[88] Qian Ge et al. “A survey of microarchitectural timing attacks and countermeasures on contemporary hardware”. In: *Journal of Cryptographic Engineering* 8.1 (2018), pp. 1–27.

[89] D. E. Geer and D. Peterson. “Failure as Design”. In: *IEEE Security Privacy* 17.6 (Nov. 2019), pp. 90–89. ISSN: 1558-4046. DOI: 10.1109/MSEC.2019.2936706.

[90] S. Ghosh and S. Sampalli. “A Survey of Security in SCADA Networks: Current Issues and Future Challenges”. In: *IEEE Access* 7 (2019), pp. 135812–135831. ISSN: 2169-3536.

[91] Brett Giller. “Implementing practical electrical glitching attacks”. In: *Black Hat Europe* (2015).

[92] Jairo Giraldo et al. “Security and privacy in cyber-physical systems: A survey of surveys”. In: *IEEE Design & Test* 34.4 (2017), pp. 7–17.

[93] Ben Gras et al. “ASLR on the Line: Practical Cache Attacks on the MMU.” In: *NDSS*. Vol. 17. 2017, p. 13.

[94] Ben Gras et al. “Translation leak-aside buffer: Defeating cache side-channel protections with TLB attacks”. In: *27th USENIX Security Symposium (USENIX Security 18)*. 2018, pp. 955–972.

[95] Daniel Gruss. “Software-based Microarchitectural Attacks”. In: (2017).

[96] Daniel Gruss, Raphael Spreitzer, and Stefan Mangard. “Cache template attacks: Automating attacks on inclusive last-level caches”. In: *24th USENIX Security Symposium (USENIX Security 15)*. 2015, pp. 897–912.

[97] Daniel Gruss et al. “Flush+ Flush: a fast and stealthy cache attack”. In: *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer. 2016, pp. 279–299.

[98] David Gullasch, Endre Bangerter, and Stephan Krenn. “Cache games bringing access-based cache attacks on AES to practice”. In: *2011 IEEE Symposium on Security and Privacy*. IEEE. 2011, pp. 490–505.

[99] Berk Gülmezoğlu et al. “A faster and more realistic flush+ reload attack on AES”. In: *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer. 2015, pp. 111–126.

[100] T. Guneysu, B. Moller, and C. Paar. “Dynamic Intellectual Property Protection for Reconfigurable Devices”. In: *2007 International Conference on Field-Programmable Technology*. Dec. 2007, pp. 169–176. DOI: 10.1109/FPT.2007.4439246.

[101] M. R. Guthaus et al. “MiBench: A free, commercially representative embedded benchmark suite”. In: *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No.01EX538)*. Dec. 2001, pp. 3–14. DOI: 10.1109/WWC.2001.990739.

[102] Seunghun Han et al. “A Bad Dream: Subverting Trusted Platform Module While You Are Sleeping”. In: *27th USENIX Security Symposium (USENIX Security 18)*. Baltimore, MD: USENIX Association, Aug. 2018, pp. 1229–1246. ISBN: 978-1-939133-04-5. URL: https://www.usenix.org/conference/usenixsecurity18/presentation/han.

[103] Houcine Hassan et al. “Special issue on: “Heterogeneous architectures for Cyber-physical systems (HACPS)””. In: *Microprocessors and Microsystems* 52 (2017), pp. 333–334. ISSN: 0141-9331. DOI: 10.1016/j.micpro.2017.03.010. URL: http://www.sciencedirect.com/science/article/pii/S0141933117301874.

[104] N. A. Hazari, F. Alsulami, and M. Niamat. “FPGA IP Obfuscation Using Ring Oscillator Physical Unclonable Function”. In: *NAECON 2018 - IEEE National Aerospace and Electronics Conference*. July 2018, pp. 105–108. DOI: 10.1109/NAECON.2018.8556746.

[105] J.L. Hennessy and D.A. Patterson. *Computer Architecture: A Quantitative Approach*. ISSN. Elsevier Science, 2017. ISBN: 9780128119068. URL: https://books.google.it/books?id=cM8mDwAAQBAJ.

[106] Mark D Hill et al. “On the Spectre and Meltdown Processor Security Vulnerabilities”. In: *IEEE Micro* 39.2 (2019), pp. 9–19.

[107] DHS (Department of Homeland Security). *Homeland security presidential directive 7: Critical infrastructure identification, prioritization, and protection*. 2003.

[108] Seokhie Hong. “Special Issue on “Side Channel Attacks””. In: *Applied Sciences* 9.9 (May 2019), p. 1881. ISSN: 2076-3417. DOI: 10.3390/app9091881. URL: http://dx.doi.org/10.3390/app9091881.

[109] Y. Hori et al. “Bitstream encryption and authentication with AES-GCM in dynamically reconfigurable systems”. In: *Proc. Int. Conf. Field Programmable Logic and Applications FPL 2008*. 2008, pp. 23–28. DOI: 10.1109/FPL.2008.4629902.

[110] White House. “Improving critical infrastructure cybersecurity”. In: *Executive Order EO-13636, Office of the Press Secretary, Washington, DC* (2013).

[111] White House. “Presidential policy directive/PPD 21–Critical infrastructure security and resilience”. In: *Washington, DC* (2013).

[112] A. Humayed et al. “Cyber-Physical Systems Security—A Survey”. In: *IEEE Internet of Things Journal* 4.6 (Dec. 2017), pp. 1802–1831. ISSN: 2372-2541.

[113] Ralf Hund, Carsten Willems, and Thorsten Holz. "Practical timing side channel attacks against kernel space ASLR". In: *2013 IEEE Symposium on Security and Privacy*. IEEE. 2013, pp. 191–205.

[114] William Hupp et al. "Module-OT: A Hardware Security Module for Operational Technology". In: *2020 IEEE Texas Power and Energy Conference (TPEC)*. IEEE. 2020, pp. 1–6.

[115] Michael Hutter and Jörn-Marc Schmidt. "The temperature side channel and heating fault attacks". In: *International Conference on Smart Card Research and Advanced Applications*. Springer. 2013, pp. 219–235.

[116] Vinay M Igure, Sean A Laughter, and Ronald D Williams. "Security issues in SCADA networks". In: *computers & security* 25.7 (2006), pp. 498–506.

[117] Mehmet Sinan Inci. "Micro-architectural Threats to Modern Computing Systems". In: (2019).

[118] Intel. *AN 556: Using the Design Security Features in Intel FPGAs*. Intel.

[119] Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar. "S $ A: A shared cache attack that works across cores and defies VM sandboxing–and its application to AES". In: *2015 IEEE Symposium on Security and Privacy*. IEEE. 2015, pp. 591–604.

[120] Gorka Irazoqui et al. "Know thy neighbor: Crypto library detection in cloud". In: *Proceedings on Privacy Enhancing Technologies* 2015.1 (2015), pp. 25–40.

[121] iResearch. *Worldwide mobile app revenues in 2014 to 2023*. URL: https://www.statista.com/statistics/269025/worldwide-mobile-app-revenue-forecast/ (visited on 2019).

[122] Himanshi Jain, D Anthony Balaraju, and Chester Rebeiro. "Spy Cartel: Parallelizing Evict+ Time-Based Cache Attacks on Last-Level Caches". In: *Journal of Hardware and Systems Security* 3.2 (2019), pp. 147–163.

[123] S. K. K. et al. "A Flexible Pay-per-Device Licensing Scheme for FPGA IP Cores". In: *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. July 2017, pp. 677–682. DOI: 10.1109/ISVLSI.2017.123.

[124] Manolis Kaliorakis et al. "MeRLiN: Exploiting Dynamic Instruction Behavior for Fast and Accurate Microarchitecture Level Reliability Assessment". In: *Proceedings of the 44th Annual International Symposium on Computer Architecture*. ISCA '17. Toronto, ON, Canada: ACM, 2017, pp. 241–254. ISBN: 978-1-4503-4892-8. DOI: 10.1145/3079856.3080225. URL: http://doi.acm.org/10.1145/3079856.3080225.

[125] A. Kandalintsev et al. "Profiling cloud applications with hardware performance counters". In: *The International Conference on Information Networking 2014 (ICOIN2014)*. Feb. 2014, pp. 52–57. DOI: 10.1109/ICOIN.2014.6799664.

[126] D. KAO, S. HSIAO, and R. TSO. "Analyzing WannaCry Ransomware Considering the Weapons and Exploits". In: *2019 21st International Conference on Advanced Communication Technology (ICACT)*. Feb. 2019, pp. 1098–1107. DOI: 10.23919/ICACT.2019.8702049.

[127] Da-Yu Kao and Shou-Ching Hsiao. "The dynamic analysis of WannaCry ransomware". In: *2018 20th International Conference on Advanced Communication Technology (ICACT)*. IEEE. 2018, pp. 159–166.

[128] R. Karam et al. "Robust bitstream protection in FPGA-based systems through low-overhead obfuscation". In: *2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. Nov. 2016, pp. 1–8. DOI: 10.1109/ReConFig.2016.7857187.

[129] Hirak Kashyap and Ricardo Chaves. "Compact and On-the-Fly Secure Dynamic Reconfiguration for Volatile FPGAs". In: *ACM Trans. Reconfigurable Technol. Syst.* 9.2 (Jan. 2016), 11:1–11:22. ISSN: 1936-7406. DOI: 10.1145/2816822. URL: http://doi.acm.org/10.1145/2816822.

[130] Mehmet Kayaalp et al. "A high-resolution side-channel attack on last-level cache". In: *Proceedings of the 53rd Annual Design Automation Conference*. 2016, pp. 1–6.

[131] S. Kent and R. Atkinson. *Security Architecture for the Internet Protocol*. RFC 2401. United States: IETF, 1998.

[132] Richard Kissel. "Glossary of Key Information Security Terms, NISTIR 7298 Revision 2". In: *Gaithersburg: National Institute of Standards and Technology, Computer Security Division, & Information Technology Laboratory, Eds. https://doi. org/10.6028/NIST. IR. 7298r2* (2013).

[133] David Klaper and Eduard Hovy. "A taxonomy and a knowledge portal for cybersecurity". In: *Proceedings of the 15th annual international conference on digital government research*. 2014, pp. 79–85.

[134] John C. Knight. "Safety Critical Systems: Challenges and Directions". In: *Proceedings of the 24th International Conference on Software Engineering*. ICSE '02. Orlando, Florida: Association for Computing Machinery, 2002, pp. 547–550. ISBN: 158113472X. DOI: 10.1145/581339.581406. URL: https://doi.org/10.1145/581339.581406.

[135] Paul Kocher, Joshua Jaffe, and Benjamin Jun. "Differential power analysis". In: *Annual International Cryptology Conference*. Springer. 1999, pp. 388–397.

[136] Paul Kocher et al. "Introduction to differential power analysis". In: *Journal of Cryptographic Engineering* 1.1 (Apr. 2011), pp. 5–27. ISSN: 2190-8516. DOI: 10.1007/s13389-011-0006-y.

[137] Paul Kocher et al. "Spectre attacks: Exploiting speculative execution". In: *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2019, pp. 1–19.

[138] Paul C Kocher. "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems". In: *Annual International Cryptology Conference*. Springer. 1996, pp. 104–113.

[139] Andrey V. Krasovsky and Ekaterina A. Maro. "Actual and Historical State of Side Channel Attacks Theory". In: *Proceedings of the 12th International Conference on Security of Information and Networks*. SIN '19. Sochi, Russia: Association for Computing Machinery, 2019. ISBN: 9781450372428. DOI: 10.1145/3357613.3357627. URL: https://doi.org/10.1145/3357613.3357627.

[140] Ralph Langner. "Stuxnet: Dissecting a cyberwarfare weapon". In: *IEEE Security & Privacy* 9.3 (2011), pp. 49–51.

[141] Donald C Latham. "Department of defense trusted computer system evaluation criteria". In: *Department of Defense* (1986).

[142] Lattice. *XP2 Family Handbook*. URL: http://www.latticesemi.com/documents/HB1004.pdf (visited on 2012).

[143] Nate Lawson. "Side-channel attacks on cryptographic software". In: *IEEE Security & Privacy* 7.6 (2009), pp. 65–68.

[144] Jay Lee, Behrad Bagheri, and Hung-An Kao. "A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems". In: *Manufacturing Letters* 3 (2015), pp. 18–23. ISSN: 2213-8463. DOI: https://doi.org/10.1016/j.mfglet.2014.12.001. URL: http://www.sciencedirect.com/science/article/pii/S221384631400025X.

[145] Austin Lesea. *IP security in FPGAs*. Xilinx Inc. Feb. 2007. URL: http://www.xilinx.com/support/documentation/white%5C_papers/wp261.pdf.

[146] Y. Li, M. Chen, and J. Wang. "Introduction to side-channel attacks and fault attacks". In: *2016 Asia-Pacific International Symposium on Electromagnetic Compatibility (APEMC)*. Vol. 01. May 2016, pp. 573–575. DOI: 10.1109/APEMC.2016.7522801.

[147] Wang Lie and Wu Feng-Yan. "Dynamic partial reconfiguration in FPGAs". In: *2009 Third International Symposium on Intelligent Information Technology Application*. Vol. 2. IEEE. 2009, pp. 445–448.

[148] Moritz Lipp et al. "Armageddon: Cache attacks on mobile devices". In: *25th USENIX Security Symposium (USENIX Security 16)*. 2016, pp. 549–564.

[149] Moritz Lipp et al. "Meltdown: Reading kernel memory from user space". In: *27th USENIX Security Symposium (USENIX Security 18)*. 2018, pp. 973–990.

[150] Fangfei Liu et al. "Last-level cache side-channel attacks are practical". In: *2015 IEEE symposium on security and privacy*. IEEE. 2015, pp. 605–622.

[151] Yang Liu et al. "Review on cyber-physical systems". In: *IEEE/CAA Journal of Automatica Sinica* 4.1 (2017), pp. 27–40.

[152] R. Maes, D. Schellekens, and I. Verbauwhede. "A Pay-per-Use Licensing Scheme for Hardware IP Cores in Recent SRAM-Based FPGAs". In: *IEEE Transactions on Information Forensics and Security* 7.1 (Feb. 2012), pp. 98–108. ISSN: 1556-6013. DOI: 10.1109/TIFS.2011.2169667.

[153] Mohamad Imad Mahaini, Rahime Belen Sağlam, and Shujun Li. "Building Taxonomies based on Human-Machine Teaming: Cyber Security as an Example". In: *14th International Conference on Availability, Reliability and Security ARES 2018)*. ACM, Aug. 2019. URL: https://dl.acm.org/doi/10.1145/3339252.3339282.

[154] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks: Revealing the secrets of smart cards*. Vol. 31. Springer Science & Business Media, 2008.

[155] J. M. Mcginthy and A. J. Michaels. "Secure Industrial Internet of Things Critical Infrastructure Node Design". In: *IEEE Internet of Things Journal* 6.5 (Oct. 2019), pp. 8021–8037. ISSN: 2372-2541. DOI: 10.1109/JIOT.2019.2903242.

[156] Stephen E McLaughlin. "On Dynamic Malware Payloads Aimed at Programmable Logic Controllers." In: *HotSec*. 2011.

[157] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography*. 1st. Boca Raton, FL, USA: CRC Press, Inc., 1996. ISBN: 0849385237.

[158] Thomas S Messerges, Ezzy A Dabbish, and Robert H Sloan. "Investigations of Power Analysis Attacks on Smartcards." In: *Smartcard* 99 (1999), pp. 151–161.

[159] E. Mezzetti et al. "High-Integrity Performance Monitoring Units in Automotive Chips for Reliable Timing V and V". In: *IEEE Micro* 38.1 (Jan. 2018), pp. 56–65. ISSN: 0272-1732. DOI: 10.1109/MM.2018.112130235.

[160] *Modbus messaging on tcp/ip implementation guide v1.0b*. Standard. Oct. 2006. URL: http://www.modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf.

[161] Daniel Moghimi et al. "TPM-FAIL: TPM meets Timing and Lattice Attacks". In: *29th USENIX Security Symposium (USENIX Security 20)*. Boston, MA: USENIX Association, Aug. 2020. URL: https://www.usenix.org/conference/usenixsecurity20/presentation/moghimi.

[162] S. Moore et al. "PAPI deployment, evaluation, and extensions". In: *2003 User Group Conference. Proceedings.* June 2003, pp. 349–353. DOI: 10.1109/DODUGC.2003.1253415.

[163] Amir Moradi et al. "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*. CCS '11. Chicago, Illinois, USA: ACM, 2011, pp. 111–124. ISBN: 978-1-4503-0948-6. DOI: 10.1145/2046707.2046722. URL: http://doi.acm.org/10.1145/2046707.2046722.

[164] Amir Moradi et al. "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*. FPGA '13. Monterey, California, USA: ACM, 2013, pp. 91–100. ISBN: 978-1-4503-1887-7. DOI: 10.1145/2435264.2435282. URL: http://doi.acm.org/10.1145/2435264.2435282.

[165] G. D. Natale, M. L. Flottes, and B. Rouzeyre. "An Integrated Validation Environment for Differential Power Analysis". In: *4th IEEE International Symposium on Electronic Design, Test and Applications (delta 2008)*. Jan. 2008, pp. 527–532. DOI: 10.1109/DELTA.2008.61.

[166] G. Naumovich and N. Memon. "Preventing piracy, reverse engineering, and tampering". In: *Computer* 36.7 (July 2003), pp. 64–71. ISSN: 0018-9162. DOI: 10.1109/MC.2003.1212692.

[167] Sajid Nazir, Shushma Patel, and Dilip Patel. "Autonomic computing meets SCADA security". In: *2017 IEEE 16th International Conference on Cognitive Informatics & Cognitive Computing (ICCI* CC)*. IEEE. 2017, pp. 498–502.

[168] Michael Neve and Jean-Pierre Seifert. "Advances on Access-Driven Cache Attacks on AES". In: *Selected Areas in Cryptography*. Ed. by Eli Biham and Amr M. Youssef. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 147–162. ISBN: 978-3-540-74462-7.

[169] Marie Nguyen and James C Hoe. "Time-shared execution of realtime computer vision pipelines by dynamic partial reconfiguration". In: *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE. 2018, pp. 230–2304.

[170] Nguyen Tran Huu Nguyen. "Repairing FPGA configuration memory errors using dynamic partial reconfiguration". In: *Ph. D. dissertation, The University of New South Wales* (2017).

[171] Jean-Baptiste Note and Éric Rannaud. "From the bitstream to the netlist". In: *ACM/SIGDA Symposium on Field Programmable Gate Arrays*. ACM New York, NY, USA, Feb. 2008, pp. 264–264. URL: http://islsm.org/~jb/debit/bitstream.pdf.

[172] T. Novak and A. Treytl. "Functional safety and system security in automation systems - a life cycle model". In: *2008 IEEE International Conference on Emerging Technologies and Factory Automation*. Sept. 2008, pp. 311–318. DOI: 10.1109/ETFA.2008.4638412.

[173] E. Novillo and P. Lu. "On-line debugging and performance monitoring with barriers". In: *Proceedings 15th International Parallel and Distributed Processing Symposium. IPDPS 2001*. Apr. 2001, 8 pp.-. DOI: 10.1109/IPDPS.2001.924959.

[174] J. Nowotsch et al. "Monitoring and WCET analysis in COTS multi-core-SoC-based mixed-criticality systems". In: *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*. Mar. 2014, pp. 1–5. DOI: 10.7873/DATE.2014.080.

[175] J. Nowotsch et al. "Multi-core Interference-Sensitive WCET Analysis Leveraging Runtime Resource Capacity Enforcement". In: *2014 26th Euromicro Conference on Real-Time Systems*. July 2014, pp. 109–118. DOI: 10.1109/ECRTS.2014.20.

[176] Sébastien Ordas, Ludovic Guillaume-Sage, and Philippe Maurine. "Electromagnetic fault injection: the curse of flip-flops". In: *Journal of Cryptographic Engineering* 7.3 (2017), pp. 183–197.

[177] Björn Osterloh et al. "Dynamic partial reconfiguration in space applications". In: *2009 NASA/ESA Conference on Adaptive Hardware and Systems*. IEEE. 2009, pp. 336–343.

[178] Dag Arne Osvik, Adi Shamir, and Eran Tromer. "Cache attacks and countermeasures: the case of AES". In: *Cryptographers' track at the RSA conference*. Springer. 2006, pp. 1–20.

[179] M. M. Parelkar and K. Gaj. "Implementation of EAX mode of operation for FPGA bitstream encryption and authentication". In: *Proc. IEEE Int Field-Programmable Technology Conf.* 2005, pp. 335–336. DOI: 10.1109/FPT.2005.1568588.

[180] Milind M. Parelkar. "Authenticated encryption in hardware". Master's thesis. George Mason University, 2005.

[181] A. Paschalis and D. Gizopoulos. "Effective software-based self-test strategies for on-line periodic testing of embedded processors". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 24.1 (Jan. 2005), pp. 88–99. ISSN: 0278-0070. DOI: 10.1109/TCAD.2004.839486.

[182] S. Paudel, M. Tauber, and I. Brandic. "Security standards taxonomy for Cloud applications in Critical Infrastructure IT". In: *8th International Conference for Internet Technology and Secured Transactions (ICITST-2013)*. Dec. 2013, pp. 645–646. DOI: 10.1109/ICITST.2013.6750282.

[183] Payment Card Industry PCI. "PCI PIN Transaction Security (PTS) Hardware Security Module (HSM) v3". In: (2016).

[184] C. Peng et al. "A Survey on Security Communication and Control for Smart Grids Under Malicious Cyber Attacks". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 49.8 (Aug. 2019), pp. 1554–1569. ISSN: 2168-2232. DOI: 10.1109/TSMC.2018.2884952.

[185] Colin Percival. *Cache missing for fun and profit*. 2005.

[186] Darshika G. Perera. "Analysis of FPGA-Based Reconfiguration Methods for Mobile and Embedded Applications". In: *Proceedings of the 12th FPGAworld Conference 2015*. Stockholm, Sweden: ACM, 2015, pp. 15–20. ISBN: 978-1-4503-3737-3.

[187] L. Pièvre-Cambacédès and M. Bouissou. "Cross-fertilization between safety and security engineering". In: *Reliability Engineering & System Safety* 110 (2013), pp. 110–126. ISSN: 0951-8320. DOI: 10.1016/j.ress.2012.09.011. URL: http://www.sciencedirect.com/science/article/pii/S0951832012001913.

[188] Ludovic Pièvre-Cambacédès and Marc Bouissou. "Modeling safety and security interdependencies with BDMP (Boolean logic Driven Markov Processes)". In: *Systems Man and Cybernetics (SMC), 2010 IEEE International Conference on*. IEEE. 2010, pp. 2852–2861.

[189] Ludovic Pièvre-Cambacédès and Claude Chaudet. "The SEMA referential framework: Avoiding ambiguities in the terms "security" and "safety"". In: *International Journal of Critical Infrastructure Protection* 3.2 (2010), pp. 55–66. ISSN: 1874-5482. DOI: 10.1016/j.ijcip.2010.06.003. URL: http://www.sciencedirect.com/science/article/pii/S1874548210000247.

[190] *Position Paper CAST-32A, Multi-core Processors*. Certification Authorities Software Team (CAST). 2016.

[191] R. Priya et al. "A survey on security attacks in electronic healthcare systems". In: *2017 International Conference on Communication and Signal Processing (ICCSP)*. Apr. 2017, pp. 0691–0694. DOI: 10.1109/ICCSP.2017.8286448.

[192] Yu Qin, Yingjun Zhang, and Wei Feng. "TICS: Trusted Industry Control System Based on Hardware Security Module". In: *International Symposium on Cyberspace Safety and Security*. Springer. 2017, pp. 485–493.

[193] J-J Quisquater. "Eddy current for magnetic analysis with active sensor". In: *Proceedings of Esmart, 2002* (2002), pp. 185–194.

[194] Jean-Jacques Quisquater and David Samyde. "Electromagnetic analysis (ema): Measures and counter-measures for smart cards". In: *International Conference on Research in Smart Cards*. Springer. 2001, pp. 200–210.

[195] Ciprian-Radu Rad et al. "Smart monitoring of potato crop: a cyber-physical system architecture model in the field of precision agriculture". In: *Agriculture and Agricultural Science Procedia* 6 (2015), pp. 73–79.

[196] R. Rajkumar et al. "Cyber-physical systems: The next computing revolution". In: *Design Automation Conference*. June 2010, pp. 731–736. DOI: 10.1145/1837274.1837461.

[197] Umesh Hodeghatta Rao and Umesha Nayak. "History of Computer Security". In: *The InfoSec Handbook: An Introduction to Information Security*. Berkeley, CA: Apress, 2014, pp. 13–25. ISBN: 978-1-4302-6383-8. DOI: 10.1007/978-1-4302-6383-8_2. URL: https://doi.org/10.1007/978-1-4302-6383-8_2.

[198] Chester Rebeiro, Debdeep Mukhopadhyay, and Sarani Bhattacharya. "Time-Driven Cache Attacks". In: *Timing Channels in Cryptography: A Micro-Architectural Perspective*. Cham: Springer International Publishing, 2015, pp. 53–70. ISBN: 978-3-319-12370-7. DOI: 10.1007/978-3-319-12370-7_4. URL: https://doi.org/10.1007/978-3-319-12370-7_4.

[199] Thomas Ristenpart et al. "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds". In: *Proceedings of the 16th ACM conference on Computer and communications security*. 2009, pp. 199–212.

[200] C. Roscian et al. "Fault Model Analysis of Laser-Induced Faults in SRAM Memory Cells". In: *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*. Aug. 2013, pp. 89–98. DOI: 10.1109/FDTC.2013.17.

[201] Ronald S Ross. *Recommended Security Controls for Federal Information Systems and Organizations [includes updates through 9/14/2009]*. Tech. rep. National Institute of Standards and Technology, 2009.

[202] B. Roux et al. "Fast and Energy-Driven Design Space Exploration for Heterogeneous Architectures". In: *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 2017, pp. 83–83.

[203] Giedre Sabaliauskaite and Aditya P Mathur. "Aligning cyber-physical system safety and security". In: *Complex Systems Design & Management Asia*. Springer, 2015, pp. 41–53.

[204] H. Saito, Y. Tomioka, and J. Kitamichi. "Proposing a Highly Reliable Real-Time Operating System for a Processor with a Fault Self-Detecting Mechanism". In: *2016 13th International Conference on Embedded Software and Systems (ICESS)*. Aug. 2016, pp. 164–169. DOI: 10.1109/ICESS.2016.9.

[205] Nagham Samir et al. "Energy-adaptive lightweight hardware security module using partial dynamic reconfiguration for energy limited internet of things applications". In: *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE. 2019, pp. 1–4.

[206] G. P. H. Sandaruwan, P. S. Ranaweera, and V. A. Oleshchuk. "PLC security and critical infrastructure protection". In: *2013 IEEE 8th International Conference on Industrial and Information Systems*. Dec. 2013, pp. 81–85. DOI: 10.1109/ICIInfS.2013.6731959.

[207] A. Savino, A. Vallero, and S. Di Carlo. "ReDO: Cross-Layer Multi-Objective Design-Exploration Framework for Efficient Soft Error Resilient Systems". In: *IEEE Transactions on Computers* (2018), pp. 1–1. ISSN: 0018-9340. DOI: 10.1109/TC.2018.2818735.

[208] Stephan van Schaik et al. "Reverse engineering hardware page table caches using side-channel attacks on the MMU". In: *VU Amsterdam* (2017).

[209] Michael N Schmitt. *Tallinn manual on the international law applicable to cyber warfare*. Cambridge University Press, 2013.

[210] Michael Schwarz. "Software-based Side-Channel Attacks and Defenses in Restricted Environments". In: (2019).

[211] M. Scott. "MIRACL-A multiprecision integer and rational arithmetic C/C++ library". In: *http://www.shamus.ie* (2003). URL: https://ci.nii.ac.jp/naid/10026809141/en/.

[212] ITU-D Secretariat. "ITU Study Group Q.22/1 Report on Best Practices for a National Approach to Cybersecurity: A Management Framework for Organizing National Cybersecurity Efforts". In: (2008).

[213] Robert Shirey. *Internet security glossary, version 2*. Tech. rep. RFC 4949, August, 2007.

[214] Tobby Simon. "Critical infrastructure and the Internet of Things". In: *Global Commission on Internet Governance Paper Series* (2017).

[215] P. Sinha et al. "Security vulnerabilities, attacks and countermeasures in wireless sensor networks at various layers of OSI reference model: A survey". In: *2017 International Conference on Signal Processing and Communication (ICSPC)*. July 2017, pp. 288–293. DOI: 10.1109/CSPC.2017.8305855.

[216] Sergei P Skorobogatov and Ross J Anderson. "Optical fault induction attacks". In: *International workshop on cryptographic hardware and embedded systems*. Springer. 2002, pp. 2–12.

[217] Raphael Spreitzer and Thomas Plos. "Cache-access pattern attack on disaligned aes t-tables". In: *International Workshop on Constructive Side-Channel Analysis and Secure Design*. Springer. 2013, pp. 200–214.

[218] Raphael Spreitzer et al. "Systematic classification of side-channel attacks: a case study for mobile devices". In: *IEEE Communications Surveys & Tutorials* 20.1 (2017), pp. 465–488.

[219] Martin Stigge et al. *Reversing CRC theory and practice*. Technical Report SAR-PR-2006-05. Humboldt University Berlin, 2006.

[220] *STM32 Nucleo-144 board Data brief*. DB2762. Rev. 6. ST Microelectronics. Mar. 2017.

[221] *STM32F427xx STM32F429xx Datasheet - production data*. DS9405. Rev. 9. ST Microelectronics. July 2016.

[222] H. Sun and Q. Ding. "The Design of the Mobile Phone Module Information Encryption System Based on FPGA". In: *2015 Fifth International Conference on Instrumentation and Measurement, Computer, Communication and Control (IMCCC)*. Sept. 2015, pp. 1343–1347. DOI: 10.1109/IMCCC.2015.288.

[223] M. Surratt et al. "Challenges of remote FPGA configuration for space applications". In: *Aerospace Conference, 2005 IEEE*. Ieee. 2005, pp. 1–9.

[224] Jakub Szefer. "Survey of microarchitectural side and covert channels, attacks, and defenses". In: *Journal of Hardware and Systems Security* 3.3 (2019), pp. 219–234.

[225] H. Tao, J. Zhou, and S. Liu. "A survey of network security situation awareness in power monitoring system". In: *2017 IEEE Conference on Energy Internet and Energy System Integration (EI2)*. Nov. 2017, pp. 1–3. DOI: 10.1109/EI2.2017.8245487.

[226] M. U. Tariq, J. Florence, and M. Wolf. "Improving the Safety and Security of Wide-Area Cyber-Physical Systems Through a Resource-Aware, Service-Oriented Development Methodology". In: *Proceedings of the IEEE* 106.1 (Jan. 2018), pp. 144–159. ISSN: 0018-9219. DOI: 10.1109/JPROC.2017.2744645.

[227] Christopher Tarnovsky. *Hacking the Smartcard Chip*. https://www.blackhat.com/html/bh-dc-10/bh-dc-10-briefings.html. 2010.

[228] Taxonomy. *Cambridge Dictionary*. Cambridge University Press. URL: https://dictionary.cambridge.org/dictionary/english/taxonomy.

[229] *TDS5000B Series Digital Phosphor Oscilloscopes - Quick Start User Manual*. 071-1355-02. Tektronix.

[230] T. Thanh et al. "Secure remote updating of bitstream in partial reconfigurable embedded systems based on FPGA". In: *2013 International Conference on Computing, Management and Telecommunications (ComManTel)*. Jan. 2013, pp. 152–156. DOI: 10.1109/ComManTel.2013.6482382.

[231] G. Theodorou et al. "Software-Based Self-Test for Small Caches in Microprocessors". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 33.12 (Dec. 2014), pp. 1991–2004. ISSN: 0278-0070. DOI: 10.1109/TCAD.2014.2363387.

[232] V. L. L. Thing and J. Wu. "Autonomous Vehicle Security: A Taxonomy of Attacks and Defences". In: *2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. Dec. 2016, pp. 164–170. DOI: 10.1109/iThings-GreenCom-CPSCom-SmartData.2016.52.

[233] S. Trimberger and J. Moore. "FPGA security: From features to capabilities to trusted systems". In: *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*. June 2014, pp. 1–4. DOI: 10.1145/2593069.2602555.

[234] Eran Tromer, Dag Arne Osvik, and Adi Shamir. "Efficient cache attacks on AES, and countermeasures". In: *Journal of Cryptology* 23.1 (2010), pp. 37–71.

[235] Trusted Computing Group. *Trusted Platform Module 1.2 Main Specification*. URL: https://trustedcomputinggroup.org/resource/tpm-main-specification/ (visited on 2019).

[236] Trusted Computing Group. *Trusted Platform Module Library Part 3: Commands (Level 00 Revision 01.38)*. URL: https://www.trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-2.0-Part-3-Commands-01.38.pdf (visited on 2019).

[237] L. Uhsadel, A. Georges, and I. Verbauwhede. "Exploiting Hardware Performance Counters". In: *2008 5th Workshop on Fault Diagnosis and Tolerance in Cryptography*. Aug. 2008, pp. 59–67. DOI: 10.1109/FDTC.2008.19.

[238] Darshana Upadhyay and Srinivas Sampalli. "SCADA (Supervisory Control and Data Acquisition) systems: Vulnerability assessment and security recommendations". In: *Computers & Security* 89 (2020), p. 101666.

[239] A. Vallero, A. Carelli, and S. Di Carlo. "Trading-off reliability and performance in FPGA-based reconfigurable heterogeneous systems". In: *2018 13th International Conference on Design & Technology of Integrated Systems In Nanoscale Era (DTIS)*. IEEE. 2018, pp. 1–6.

[240] A. Vallero, D. Gizopoulos, and S. Di Carlo. "SIFI: AMD southern islands GPU microarchitectural level fault injector". In: *2017 IEEE 23rd International Symposium on On-Line Testing and Robust System Design (IOLTS)*. 2017, pp. 138–144.

[241] A. Vallero et al. "Bayesian models for early cross-layer reliability analysis and design space exploration". In: *2019 IEEE 25th International Symposium on On-Line Testing and Robust System Design (IOLTS)*. IEEE. 2019, pp. 143–146.

[242] A. Vallero et al. "Cross-layer system reliability assessment framework for hardware faults". In: *2016 IEEE International Test Conference (ITC)*. Nov. 2016, pp. 1–10. DOI: 10.1109/TEST.2016.7805863.

[243] A. Vallero et al. "SyRA: Early System Reliability Analysis for Cross-Layer Soft Errors Resilience in Memory Arrays of Microprocessor Systems". In: *IEEE Transactions on Computers* 68.5 (2019), pp. 765–783.

[244] Alessandro Vallero et al. "Cross-layer reliability evaluation, moving from the hardware architecture to the system level: A CLERECO EU project overview". In: *Microprocessors and Microsystems* 39.8 (2015), pp. 1204–1214.

[245] J. G. J. van Woudenberg, M. F. Witteman, and F. Menarini. "Practical Optical Fault Injection on Secure Microcontrollers". In: *2011 Workshop on Fault Diagnosis and Tolerance in Cryptography*. Sept. 2011, pp. 91–99. DOI: 10.1109/FDTC.2011.12.

[246] A. Varriale et al. "SEcube (TM): Data at rest and data in motion protection". In: *Proceedings of the International Conference on Security and Management (SAM)*. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp). 2016, p. 138.

[247] Daimeng Wang et al. "Unveiling your keystrokes: A Cache-based Side-channel Attack on Graphics Libraries." In: *NDSS*. 2019.

[248] Lihui Wang and Xi Vincent Wang. *Cloud-Based Cyber-Physical Systems in Manufacturing*. Springer, 2018.

[249] X. Wang et al. "ConFirm: Detecting firmware modifications in embedded systems using Hardware Performance Counters". In: *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. Nov. 2015, pp. 544–551. DOI: 10.1109/ICCAD.2015.7372617.

[250] Michael Weiß, Benedikt Heinz, and Frederic Stumpf. "A Cache Timing Attack on AES in Virtualization Environments". In: *Financial Cryptography and Data Security*. Ed. by Angelos D. Keromytis. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 314–328. ISBN: 978-3-642-32946-3.

[251] Kyle Wilkinson. *Using Encryption to Secure a 7 Series FPGA Bitstream*. Xilinx.

[252] Remigiusz Wiśniewski et al. "Dynamic partial reconfiguration of concurrent control systems implemented in FPGA devices". In: *IEEE Transactions on Industrial Informatics* 13.4 (2017), pp. 1734–1741.

[253] M. Wolf and D. Serpanos. "Safety and Security in Cyber-Physical Systems and Internet-of-Things Systems". In: *Proceedings of the IEEE* 106.1 (Jan. 2018), pp. 9–20. ISSN: 0018-9219. DOI: 10.1109/JPROC.2017.2781198.

[254] Thomas Wollinger, Jorge Guajardo, and Christof Paar. "Security on FPGAs: State-of-the-art implementations and attacks". In: *ACM Trans. Embed. Comput. Syst.* 3.3 (2004), pp. 534–574. ISSN: 1539-9087. DOI: 10.1145/1015047.1015052.

[255] Yubin Xia et al. "CFIMon: Detecting violation of control flow integrity using performance counters". In: *Dependable Systems and Networks (DSN), 2012 42nd Annual IEEE/IFIP International Conference on*. IEEE. 2012, pp. 1–12.

[256] Xilinx. *Lock Your Designs with the Virtex-4 Security Solution*. URL: www.xilinx.com/publications/xcellonline/%20xcell_52/xc_pdf/xc_v4security52.pdf (visited on 2012).

[257] Xilinx Inc. *UG191: Virtex-5 configuration user guide*. URL: http://www.xilinx.com/support/documentation/user%5C_guides/ug191.pdf (visited on 2012).

[258] Yuval Yarom and Katrina Falkner. "FLUSH+ RELOAD: a high resolution, low noise, L3 cache side-channel attack". In: *23rd USENIX Security Symposium (USENIX Security 14)*. 2014, pp. 719–732.

[259] C. Yilmaz. "Using Hardware Performance Counters for Fault Localization". In: *2010 Second International Conference on Advances in System Testing and Validation Lifecycle*. Aug. 2010, pp. 87–92. DOI: 10.1109/VALID.2010.8.

[260] C. Yilmaz, A. Paradkar, and C. Williams. "Time will tell". In: *2008 ACM/IEEE 30th International Conference on Software Engineering*. May 2008, pp. 81–90. DOI: 10.1145/1368088.1368100.

[261] A. S. Zeineddini and K. Gaj. "Secure partial reconfiguration of FPGAs". In: *Proc. IEEE Int Field-Programmable Technology Conf.* 2005, pp. 155–162. DOI: 10.1109/FPT.2005.1568540.

[262] J. Zhang et al. “A PUF-FSM Binding Scheme for FPGA IP Protection and Pay-Per-Device Licensing”. In: *IEEE Transactions on Information Forensics and Security* 10.6 (June 2015), pp. 1137–1150. ISSN: 1556-6013. DOI: 10.1109/TIFS.2015.2400413.

[263] Junqi Zhang and Vijay Varadharajan. “Wireless sensor network key management survey and taxonomy”. In: *Journal of network and computer applications* 33.2 (2010), pp. 63–75.

[264] L. Zhang and C. Chang. “Public key protocol for usage-based licensing of FPGA IP cores”. In: *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*. May 2015, pp. 25–28. DOI: 10.1109/ISCAS.2015.7168561.

[265] Yinqian Zhang et al. “Cross-VM side channels and their use to extract private keys”. In: *Proceedings of the 2012 ACM conference on Computer and communications security*. 2012, pp. 305–316.