Hardware Security Knowledge Area Version 1.0.1

Ingrid Verbauwhede | KU Leuven

EDITOR

Andrew Martin | Oxford University

George Danezis | University College London

REVIEWERS

Srinivas Devadas | Massachusetts Intitute of Technology

Paul England | Microsoft

Elisabeth Oswald | University of Bristol

Mark Ryan | University of Birmingham

COPYRIGHT

The CyBOK project would like to understand how the CyBOK is being used and its uptake. The project would like organisations using, or intending to use, CyBOK for the purposes of education, training, course development, professional development etc. to contact it at **contact@cybok.org** to let the project know how they are using CyBOK.

Version 1.0.1 is a stable public release of the Hardware Security Knowledge Area.

CHANGELOG

Version date	Version number	Changes made
July 2021	1.0.1	Updated copyright statement; amended “issue” to “version”; amended typos
October 2019	1.0	

INTRODUCTION

Hardware security covers a broad range of topics from trusted computing to Trojan circuits. To classify these topics we follow the different hardware abstraction layers as introduced by the Y-chart of Gajski & Kuhn. The different layers of the hardware design process will be introduced in section 1. It is linked with the important concept of a root of trust and associated threat models in the context of hardware security. Next follows section 2 on measuring and evaluating hardware security. The next sections gradually reduce the abstraction level. Section 3 describes secure platforms, i.e. a complete system or system-on-chip as trusted computing base. Next section 4 covers hardware support for software security: what features should a programmable processor include to support software security. This section is closely related to the Software Security CyBOK Knowledge Area [1]. Register transfer level is the next abstraction level down, covered in section 5. Focus at this level is typically the efficient and secure implementation of cryptographic algorithms so that they can be mapped on ASIC or FPGA. This section is closely related to the Cryptography CyBOK Knowledge Area [2]. All implementations also need protection against physical attacks, most importantly against side-channel and fault attacks. Physical attacks and countermeasures are described in section 6. Section 7 describes entropy sources at the lowest abstraction level, close to CMOS technology. It includes the design of random numbers generators and physically unclonable functions. The last technical section describes aspects related to the hardware design process itself. This chapter ends with the conclusion and an outlook on hardware security.

1 HARDWARE DESIGN CYCLE AND ITS LINK TO HARDWARE SECURITY

Hardware security is a very broad topic and many topics fall under its umbrella. In this section, these seemingly unrelated topics are grouped and ordered according to the design levels of abstraction as introduced by the Y-chart of Gajski & Kuhn [3]. While Gajski & Kuhn propose a general approach to hardware design, in this chapter it is applied to the security aspects of hardware design and it is linked to threat models and the associated root of trust.

1.1 Short background on the hardware design process

Design abstraction layers are introduced in hardware design to reduce the complexity of the design. As indicated in 1, the lowest abstraction level a designer considers are individual transistors at the center of the figure. These transistors are composed together to form basic logic gates, such as NAND, NOR gates or flip-flops, called the logic level. Going one abstraction layer up, at register transfer level gates are grouped together to form modules, registers, ALU's, etc, and their operation is synchronized by a clock. These modules are then composed to form processors, specified by instruction sets, upon which applications and algorithms can be implemented.

By going up in the abstraction layers, details of underlying layers are hidden. This reduces design complexity at higher abstraction layers. The abstraction layers are represented by concentric circles in figure 1. Upon these circles, the Y-chart of Gajski & Kuhn introduces 3

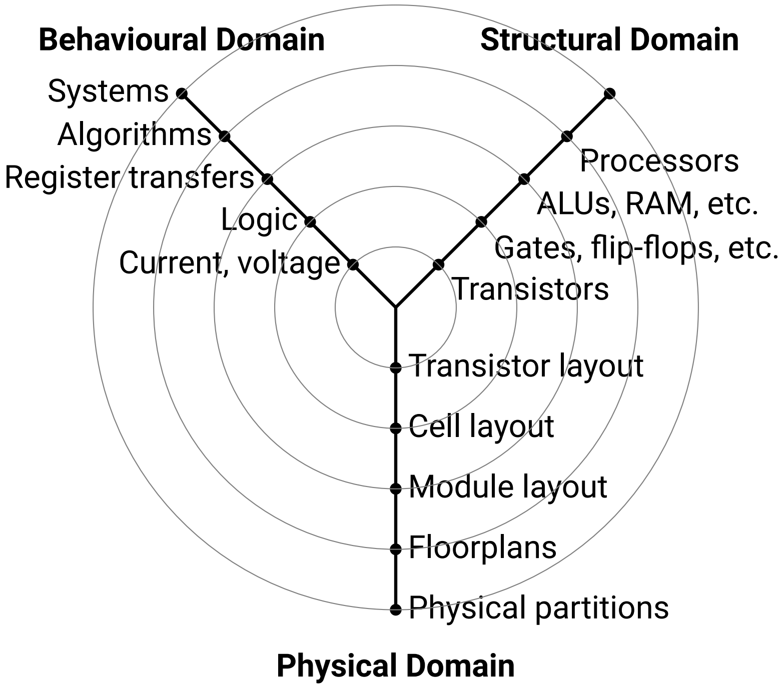

Figure 1: Gajski-Kuhn Y-chart

design activities, represented by three axes: a behavioral axis, describing the behavior or *what* needs to be implemented (aka specifications), a structural axis describing *how* something is implemented and a physical axis, how the layouts are composed together at gate, module, chip, board level. An actual design activity is a 'walk' through this design space. Typically one starts with the specifications at the top of the behavioral domain. These specifications (=what) are decomposed in components at the same level of abstraction (=how) moving from the behavioral axis to the structural axis. A structural component at one abstraction level becomes a behavioral component at one level down.

As an example of a walk through the design space: Assume a hardware designer is requested to implement a light-weight, low power security protocol for an Internet of Things (IoT) device. This designer will only receive specifications on what needs to be designed: a security protocol aims at providing confidentiality and integrity (= what) and a set of cryptographic algorithms (= components) to support the protocol. The crypto-algorithms are provided as a behavioral specification to the hardware designer, who has the choice of implementing it as a dedicated co-processor, as an assembly program, or support it with a set of custom instructions. Depending on costs and volumes, a choice of a target CMOS technology or an FPGA platform is made. This behavioral level will be translated into a more detailed register-transfer level description (e.g. VHDL or Verilog). At the Register Transfer Level (RTL), decisions need to be made if this will be a parallel or sequential version, a dedicated or programmable design, with or without countermeasures against side-channel and fault attacks, etc.

Essential for the division in design abstraction layers, is the creation of models on how components behave. E.g. to simulate the throughput or energy consumption of a arithmetic unit, quality models of the underlying gates need to be available. Similarly, the Instruction Set Architecture is a model of a processor available to the programmer.

1.2 Root of trust

In the context of security, a root of trust is a model of an underlying component for the purpose of security evaluation. According to Anderson [4]: ”A root of trust is a component used to realize a security function, upon which a designer relies but of which the trustworthiness can not be explicitly verified.” The designer uses one or multiple components to construct a security function, which then defines the trusted computing base. It is defined by the trusted computing group as follows: *“An entity can be trusted if it always behaves in the expected manner for the intended purpose.”* [5].

E.g. for an application developer, a Trusted Platform Module (TPM) or a Subscriber Identification Module (SIM) are a root of trust which the developer uses to construct a security application. For the TPM designer, the TPM is composition of smaller components which are composed together to provide security functionality. At the lowest hardware abstraction layers, basic roots of trust are the secure storage of the key in memory or the quality of the True Random Number Generator.

Hardware security is used as an enabler for software and system security. For this reason, hardware provides basic security services such as secure storage, isolation or attestation. The software or system considers the hardware as the trusted computing base. And thus from a systems or application view point, hardware has to behave as a trusted component. However, the hardware implementation can violate the trust assumption. E.g. Trojan circuits or side-channel attacks could leak the key or other sensitive data to an attacker. Hence, hardware itself also needs security. Moreover hardware needs security at all abstraction layers. Therefore, at every abstraction layer, a threat model and associated trust assumptions need to be made. An alternative definition for a root of trust in the context of design abstraction layers is therefore: “A root of trust is a component at a lower abstraction layer, upon which the system relies for its security. Its trustworthiness can either not be verified, or it is verified at a lower hardware design abstraction layer. ”

1.3 Threat model

A threat model is associated with each root of trust. When using a root of trust, it is assumed that the threat model is not violated. This means that the threat model is also linked to the hardware abstraction layers. If we consider a root of trust at a particular abstraction layer, then all components that constitute this root of trust, are also considered trusted.

Example 1: security protocols assume that the secret key is securely stored and not accessible to the attacker. The root of trust, upon which the protocol relies, is the availability of secure memory to guard this key. For the protocol designer, this secure memory is a black box. The hardware designer has to decompose this requirement for a secure memory into a set of requirements at a lower abstraction layer. What type of memory will be used? On which busses will the key travel? Which other hardware components or software have access to the storage? Can there be side-channel leaks?

Example 2: It is during this translation of higher abstraction layer requirements from protocol or security application developers into lower abstraction layers for the hardware designers that many security vulnerabilities occur. Implementations of cryptographic algorithms used to be considered black boxes to the attacker: only inputs/outputs at the algorithm level are available to mount mostly mathematical cryptanalysis attacks. However, with the appearance of side-channel attacks (see section 6) this black box assumption no longer holds. Taking side-channel leakage into account the attacker has the algorithm level information as well as the extra timing, power, electro-magnetic information as observable from the outside of the chip. Thus the attacker model moves from black box to gray box. It is still assumed that the attacker does not know the details of the internals, e.g. the contents of the key registers.

Example 3: for programmable processors, the model between hardware and software is traditionally considered the Instruction Set Architecture (ISA). The ISA is what is visible to the software programmer and the implementation of the ISA is left to the hardware designer. The ISA used to be considered the trust boundary for the software designer. Yet, with the discovery of micro-architectural side-channel attacks, such as Spectre, Meltdown, Foreshadow, this ISA model is no longer a black box, as also micro-architectural information and leakage are available to the attacker [6].

1.4 Root of trust, threat model and hardware design abstraction layers

The decomposition in abstraction layers, in combination with Electronic Design Automation (EDA) tools, is one of the main reasons that the exponential growth of Moore's law was sustainable in the past decades and it still is. This approach works well when optimizing for performance, area, energy or power consumption. Yet for hardware security, no such general decomposition exists.

In this chapter, we propose to organise the different hardware security topics, their associated threat models and root of trust according to the hardware design abstraction layers, as there is no known other general body of knowledge available to organize the topics. This organization has the advantage that it can be used to identify the state of the art on different subtopics of hardware security. As an example, in the specific context of hardware implementations of cryptographic algorithms, the state of the art is well advanced and robust countermeasures exist to protect cryptographic implementations against a wide range of side-channel attacks, as shown in detail in section 5. Yet in the context of general processor security, e.g. to isolate process related data or to provide secure execution, new security hazards continue to be discovered on a regular basis.

In an attempt to order the topics, table 1 summarizes this organization. The different abstraction layers are identified (first column) from a hardware perspective. The highest level (system and software) sits on top of the hardware platform. E.g. a system designer assumes that a secure platform is available. Thus the secure platform is the root of trust, providing security functionality. The second column describes the functionality provided by the root of trust. The third column describes how this functionality might be implemented. E.g. at the highest abstraction layer this might be by providing a Trusted Execution Module or a secure element, etc. The fourth column describes the threat models and attack categories at that abstraction layer. E.g. at system level, the system designer assumes that they will receive a module that provides isolation, integrity, attestation, etc. The last column describes typical design activities at this particular design abstraction layer.

This exercise is repeated for each abstraction layer and described in detail in each of the following sections.

At the processor level, one can distinguish general purpose programmable processors and domain specific processors. General purpose processors should support a wide range of applications, which unfortunately typically include software vulnerabilities. Hardware features are added to address these software vulnerabilities, such as a shadow stack or measures

Abstraction level	Root of trust - functionality	Structural (how) - examples	Example Threats	Typical HW design activities
System and application	Secure platforms	e.g. Trusted Execution (Trustzone, SGX, TEE), HSM, Secure Element	to support isolation, integrity, attestation, . . .	security application development
Processor	general purpose	e.g. shadow stack	SW vulnerabilities	ISA, HW/SW co-design
Processor	domain specific	Crypto specific RTL	Timing attacks	Constant number of clock cycles
Register Transfer	Crypto specific	Building blocks,	Side Channel Attack,	Logic synthesis
Logic	Resistance to SCA, Power, EM, fault	Masking, Circuit styles	Side Channel attack, fault	FPGA tools, standard cell design
Circuit and technology	Source of entropy	TRNG, PUF, Secure SRAM	Temperature, glitches	SPICE simulations
Physical	Tamper Resistance	Shields, sensors	Probing, heating	Layout activities

Table 1: Design abstraction layers linked to threat models, root of trust and design activities

to support hardware control flow integrity. Domain specific processors typically focus on a limited functionality. They are typically developed as co-processors in larger systems-on-chip. Typical examples are co-processors to support public key or secret key cryptographic algorithms. Time at the processor level is typically measured in instruction cycles.

Both general purpose and domain specific processors are composed together from computational units, multipliers and ALU's, memory and interconnect. These modules are typically described at the register transfer level: constant-time and resistance against side-channel attacks become the focus. Time at this level is typically measured in clock cycles.

Multipliers, ALU's, memories, interconnect and bus infrastructure are created from gates and flip-flops at the logic level. At this design abstraction level, focus is on leakage through physical side-channels, power, electro-magnetic, and fault attacks. Time is typically measured in absolute time (nsec) based on the available standard cell libraries or FPGA platforms.

The design of entropy sources requires knowledge and insights into the behavior of transistors and the underlying Complementary Metal-Oxide-Semiconductor (CMOS) technology.The design of these hardware security primitives is therefore positioned at the circuit and transistor level. Similarly the design of sensors and shields against physical tampering require insight into the technology. At the circuit and technology level it is measured in absolute time, e.g. nsec delay or GHz clock frequency.

The table 1 does not aim to be complete. The idea is to illustrate each abstraction layer with an example. In the next sections, the hardware security goals and their associated threat models will be discussed in detail in relation to and relevance for each abstraction layer.

2 MEASURING HARDWARE SECURITY

Depending on the commercial application domain, several industrial and government organizations have issued standards or evaluation procedures. The most well known ones are the FIPS 140-2 (and the older FIPS 140-1), the Common Criteria (CC) evaluation and in the financial world the EMVCO. FIPS 140-2 mostly focuses on the implementation security of cryptographic algorithms. Common Criteria are applicable to IT security in general.

2.1 FIPS140-2

FIPS140-2 is a US NIST standard used for the evaluation of cryptographic modules. FIPS140-2 defines security levels from 1 to 4 (1 being the lowest). The following gives a description of the four levels from a physical hardware security point of view. Next to the physical requirements, there are also roles, services and authentication requirements (for more details see [7] and other KAs).

Security level 1 only requires than an approved cryptographic algorithm be used, e.g. AES or SHA-3, but does not impose physical security requirements. Hence a software implementation could meet level 1. Level 2 requires a first level of tamper evidence. Level 3 also requires the tamper evidence, but on top requires tamper resistance.

NIST defines tampering as an intentional but unauthorized act resulting in the modification of a system, components of systems, its intended behavior, or data, [8].

Tamper evidence means that there is a proof or testimony that tampering with a hardware module has happened. E.g. a broken seal indicates that a device was opened. A light sensor might observe that the lid of a chip package was lifted.

Tamper resistance means that on top of tamper evidence, protection mechanisms are added to the device. E.g. by extra coating or dense metal layers, it is difficult to probe the key registers.

Level 4 increases the requirements such that the cryptographic module can operate in physically unprotected environments. In this context, the physical side-channel attacks pose an important threat. If any of these physical components depend on sensitive data being processed, information is leaked. Since the device is under normal operation, a classic tamper evidence mechanism will not realize that the device is under attack. See later in section 6.

2.2 Common criteria and EMVCo

"Common Criteria for information technology security evaluation" is an international standard for IT product security (ISO/IEC 15408), in short known as Common Criteria (CC). CC is a very generic procedure applicable to the security evaluation of IT products. Several parties are involved in this procedure. The customer will define a set of security specifications for its product. The manufacturer will design a product according to these specifications. An independent evaluation lab will verify if the product fulfills the claims made in the security requirements. Certification bodies will issue a certification that the procedure was correctly followed and that the evaluation lab indeed confirmed the claims made. The set of security specifications are collected in a so-called protection profile.

Depending on the amount of effort put into the security evaluation, the CC defines different Evaluation Assurance Levels (EALs). It ranges from basic functional testing, corresponding to EAL1, to formally verified design and tested, corresponding to the highest level EAL7. CC further subdivides the process of evaluation into several classes, where most of the classes verify the conformity of the device under test. The 5th class (AVA) deals with the actual vulnerability assessment. It is the most important class from a hardware security viewpoint as it searches for vulnerabilities and associated tests. It will assign a rating on the difficulty to execute the test, called the identification, and the possible benefit an attacker can gain from the penetration, called the exploitation. The difficulty is a function of the time required to perform the attack, the expertise of the attacker from layman to multiple experts, how much knowledge of the device is required from simple public information to detailed hardware source code, the number of samples required, and the cost and availability of equipment to perform the attack, etc. A high difficulty level will result in a high score and a high level of the AVA class. The highest score one can obtain is an AVA level of 5, which is required to obtain a top EAL score.

Its usage is well established in the field of smartcards and secure elements as they are used in telecom, financial, government ID's applications. It is also used in the field of Hardware Security Modules, Trusted Platform Modules and some more [9]. For certain classes of applications minimum sets of requirements are defined into protection profiles. There exists protection profiles for Trusted Platform Module (TPM), Javacards, Biometric passports, SIM cards, secure elements, etc.

Since certification comes from one body, there exist agreements between countries so that the certifications in one country are recognized in other countries. As an exception EMVCo is a private organization to set the specifications for worldwide interoperability of payment transactions. It has its own certification procedure similar to CC.

Please note that the main purpose of a common criteria evaluation is to verify that an IT product delivers the claims promised in the profile. It does not mean that there are no vulnerabilities left. A good introduction to the topic can be found in [10] and a list of certified products on [9].

2.3 SESIP: Security Evaluation Standard for IoT Platforms

In the context of IoT security evaluation, a recent initiative is the SESIP Security Evaluation scheme [11], currently at version 1.2. IoT devices are typically small, light-weight 'things', with limited accessibility via internet. Several levels of threat model for IoT are possible: from only remote internet access, over various remote software attack options, to also physical attack resistance. A comprehensive set of security functional requirements are defined: identification and attestation, product lifecycle, secure communication, software and physical attack resistance, cryptographic functionality including random number generation, and some compliance functionality to e.g. provide secure encrypted storage or provide reliable time. Similar to Common Criteria, SESIP provides several levels of assurance. Level 1 is the lowest level and consists of a self-assessment. The highest level of SESIP consists of a full CC evaluation similar to smart cards or secure elements. The levels in between cover from a black box penetration testing over white box penetration testing with or without time limitations.

3 SECURE PLATFORMS

This section describes the goals and the state-of-the-art in secure platforms. At this high level of abstraction the system designer receives a complete chip or board as trusted computing base. The system designers assume that the trusted root delivers a set of cryptographic functions, protected by the hardware and software inside the physical enclosure. Common to these platforms is that they are stand-alone pieces of silicon with a strict access policy. Depending on the provided functionality, the hardware tamper resistance and protection levels, and the communication interface, these secure platforms are used in different application fields (automotive, financial, telecom). Three important platforms are the Hardware Security Module (HSM), the Subscriber Identification Module or SIM and the Trusted Platform Module (TPM). These are briefly described next.

3.1 HSM Hardware Security Module

A HSM module will typically provide cryptographic operations, e.g. a set of public key and secret key algorithms, together with secure key management including secure generation, storage and deletion of keys. Essential to HSM's is that these operations occur in a hardened and tamper resistant environment. A TRNG and a notion of a real-time clock are usually also included. HSM's are mostly used in server back-end systems to manage keys or payment systems, e.g. in banking systems.

A HSM is used as a co-processor, attached to a host system. Its architecture typically includes a micro-processor/micro-controller, a set of crypto co-processors, secure volatile and non-volatile memory, TRNG, real-time clock, and I/O. The operations occur typically inside a tamper resistant casing. In previous generations, inside the casing multiple components reside on one board.

Recently, in some application domains, such as automotive, HSM functionality is no longer provided as a stand-alone module but is now integrated as a secure co-processor in a larger System on Chip (SoC). Indeed Moore's law enables higher integration into one SoC. What exactly is covered under HSM functionality depends on the application domain. Therefore, compliance with security levels is also evaluated by specialized independent evaluation labs according to specific protection profiles.

3.2 Secure Element and Smartcard

Similar to an HSM, a Secure Element and a smart card provide a set of cryptographic algorithms, public key, secret key, HMAC, etc. together with secure key storage, generation and deletion. The main difference with an HSM are cost, size, and form factor. They are typically implemented as one single integrated circuit and have a much smaller form factor from around 50 cm^2 to less than 1 cm^2. The main difference between a smart card and a secure element sits in the form factor and the different markets they address. Secure elements are a more generic term, while smart cards have the very specific form factor of a banking card. They are produced in large volumes and need to be very cheap as they are used for SIM cards in cell phones and smart phones. They are also used in banking cards, pay-TV systems access cards, national identity cards and passports, and recently in IOT devices, vehicular systems and so on. Tamper resistance and physical protection are essential to secure elements. They are a clear instance of what in a computer architecture domain are called 'domain specific processors'. Specific protection profiles exist depending the application domain: financial, automotive, pay-TV, etc.

A typical embedded secure element is one integrated circuit with no external components. It consists of a small micro-controller with cryptographic co-processors, secure volatile and non-volatile storage, TRNG, etc. I/O is usually limited, through a specific set of pins, or through a NFC wireless connection. Building a secure element is a challenge for a hardware designer, as one needs to combine security with non-security requirements of embedded circuits: small form factor (no external memory), low power and/or low energy consumption in combination with tamper resistance and resistance against physical attacks, such as side-channel and fault attacks (see section 6).

3.3 Trusted Platform Module (TPM)

The TPM module has been defined by the Trusted Computing Group (TCG), an industry association, to provide specific security functions to the Personal Computer (PC) platform. More specifically, the TPM is a root of trust embedded on the PC platform, so that PC+TPM platform can identify itself and its current configuration and running software [5]. The TPM provides three specific roots of trust: the Root of Trust for Measurement (RTM), the Root of Trust for Storage (RTS), the Root of Trust for Reporting (RTR). Besides these three basic functions, other functionality of TPMs is being used: access to specific cryptographic functions, secure key storage, support for secure login, etc.

The TPM is implemented as a separate security module, much like a secure element but with a specific bus interface to a PC platform, e.g. through the LPC or I^{2}C bus interface. Its architecture at minimum consists of an embedded micro-controller, several crypto coprocessors, secure volatile and non-volatile storage for root keys and a high quality true random number generator. It includes hardware engines for hash functions (SHA1 and SHA256), public key (RSA and ECC), secret key (AES) and HMAC calculations. Since a TPM is a separate module, physical protection and tamper resistance is essential for security. Next to its main scope of integrity protection, TPM also has applications in disk encryption, digital rights management, etc.

The most recent TPM2.0 version broadens the application scope from PC oriented to also supporting networking, embedded, automotive, IoT, and so on. It also provides a more flexible approach in the functionality included. Four types of TPM are identified: the dedicated integrated circuit 'discrete element' TPM provides the highest security level. One step lower in protection level is the 'integrated TPM' as an IP module in a larger SoC. The lowest levels of protection are provided by the firmware and software TPM.

The adoption of TPMs has evolved differently from what was originally the focus of the TCG. Originally, the main focus was the support of a secure boot and the associated software stack, so that a complete measurement of the software installed could be made. The problem is that the complexity of this complete software base grows too quickly, making it too difficult to measure completely all variations in valid configurations. Thus TPMs are less used to protect a complete software stack up to the higher layers of software. Still most new PCs now have TPMs but they are used to protect the encryption keys, avoid firmware roll-back, and assist the boot process in general.

Starting from the original TPM, the Trusted Computing Group has broadened its scope and now has working groups on many different application, such as cloud, embedded systems, IoT, mobile, network equipment, and so on, [12].

4 HARDWARE SUPPORT FOR SOFTWARE SECURITY AT ARCHITECTURE LEVEL

At the secure platform level, the complete module, i.e. hardware and its enclosed embedded software, are part of the trusted computing base. One level down on the abstraction layers, we make the assumption that all hardware is trusted, while software is no longer trusted. Indeed, software vulnerabilities are a major source of security weaknesses (see the Software Security CyBOK Knowledge Area [1]). To prevent the exploitation or to mitigate the effects of software vulnerabilities, a large variety of hardware modifications/additions to the processor architecture have been proposed in literature and have been included in commercial processors. We call this abstraction layer the hardware/software boundary: hardware forms the trust boundary, while software is no longer trusted. These security additions to the hardware typically have a cost in extra area and loss in performance.

The most important security objectives at this design abstraction level are to support protection, isolation and attestation for the software running on a processor platform [13], [14], [15].

- Protection: ”A set of mechanisms for ensuring that multiple processes sharing the processor, memory, or I/O devices cannot interfere, intentionally or unintentionally, with one another by reading or writing each others’ data. These mechanisms also isolate the operating system from the user process” [13]. In a traditional computer architecture, usually the OS kernel is part of the Trusted Computing Base (TCB), but the rest of the software is not.
- With isolation, a hardware mechanism is added that controls access to pieces of software and associated data. Isolation separates two parties: a software module might need protection from the surrounding software is one case. So, a Protected Model Architecture (PMA) provides a hardware guarantee that a piece of software runs unhindered from unwanted outside influences. The opposite case, if we want to limit the effects of possibly tainted software to its environment, it will be sandboxed or be put into a ‘compartment.’ Protected Module Architectures are a hardware only solution: the OS is not part of the TCB. More details are described in section 4.4
- With attestation, there is hardware support to demonstrate to a third party that the system, e.g. the code installed and/or running on a processor, is in a particular state. Attestation can be local or remote. Local attestation means that one software module can attest its state to another one on the same compute platform. Remote attestation means that a third party, outside the compute platform can get some guarantee about the state of a processor.

In the context of general purpose computing, Virtual Machines (VMs) and Hypervisors have been introduced to support multiple operating systems on one physical processor. This sharing of resources improves efficiency and reuse. It can however only be realized by a secure and efficient sharing of physical memory: virtual machines should only be allowed to use the portions of physical memory assigned to it. The organization and details of virtual memory are out of scope of hardware security and part of the Operating Systems & Virtualisation CyBOK Knowledge Area [16]. The hardware supports protection by providing privileged instructions, control and status registers and sometimes support for multiple parallel threads.

In the context of embedded micro-controllers, with no operating system, and only one application, the hardware support could be limited to only machine level support. Memory protection could be added as an optional hardware module to the processor.

Other more advanced security objectives to support software security might include:

- *Sealed storage* is the process of wrapping code and/or data with certain configuration, process or status values. Only under the correct configuration (e.g. program counter value, nonce, secret key, etc.) can the data be unsealed. *Dynamic root of trust* in combination with a *late launch* guarantees that even if the processor starts from an unknown state, it can enter a fixed known piece of code and known state. This typically requires special instructions to enter and exit the protected partition.
- *Memory protection* refers to the protection of data when it travels between the processor unit and the on-chip or off-chip memory. It protects against bus snooping or side-channel attacks or more active fault injection attacks.
- *Control flow integrity* is a security mechanism to prevent malware attacks from redirecting the flow of execution of a program. In hardware, the control flow of the program is compared on-the-fly at runtime with the expected control flow of the program.
- *Information flow analysis* is a security mechanism to follow the flow of sensitive data while it travels through the different components, from memory to cache over multiple busses into register files and processing units and back. This is important in the context of micro-architectural and physical side-channel attacks.

In the next subsections a representative set of hardware approaches to address the above software security challenges are presented. Some hardware techniques address multiple security objectives. Some are large complex approaches, others are simple dedicated hardware features.

As a side note: a large body of knowledge on software-only approaches is available in literature. Mostly, they offer a weaker level of security as they are not rooted in a hardware root of trust. E.g. for control flow integrity, software-only approaches might instruct the software code to check branches or jumps, while hardware support might calculate MACs on the fly and compare these to stored associated MACs.

4.1 Trusted Execution Environment (TEE)

TEE was originally an initiative of Global Platform, a consortium of companies, to standardize a part of the processor as a trusted secure part. TEE has since evolved and covers in general the hardware modifications made to processors to provide isolation and attestation to software applications. There is a large body of knowledge both from the industrial side as well as from the academic side.

TEE is a concept that provides a secure area of the main processor "to provide end-to-end security by protecting the execution of authenticated code, confidentiality, authenticity, privacy, system integrity and data access rights" [17]. It is important that the TEE is isolated from the so-called Rich Execution Environment (REE), which includes the untrusted OS. The reasoning behind this split is that it is impossible to guarantee secure execution and to avoid malware in the normal world due to the complexity of the OS and all other applications running there. The rich resources are accessible from the TEE, while the opposite is not possible. Global Platform does not specify the specifics on how these security properties should be implemented. Three main hardware options are suggested. Option 1 assumes that every processor component on the IC can be split into a trusted and a rich part, i.e. the processor core, the crypto accelerators, the volatile and non-volatile memory are all split. Option 2 assumes that there is a separate secure co-processor area on the SoC with a well-defined hardware interface to the rest of the SoC. Option 3 assumes a dedicated off-chip secure co-processor, much like a secure element.

Global Platform defines also a Common Criteria based protection profile (see section 2.2) for the TEE. It assumes that the package of the integrated circuit is a black box [17] and thus secure storage is assumed by the fact that the secure asset remains inside the SoC. It follows the procedures of common criteria assurance package EAL2 with some extra features. It pays extra attention to the evaluation of the random number generator and the concept of monotonic increasing time.

4.2 IBM 4758 Secure coprocessor

An early example, even before the appearance of the TEE of Global Platform is the IBM 4758 secure processor. Physical hardware security was essential for this processor: it contained a board with a general purpose processor, DRAM, separate battery backed-DRAM, Flash ROM, crypto accelerator (for DES), a random number generator and more. All of these components were enclosed in a box with tamper resistant and tamper evidence measures. It was certified to FIPS 140-1, level 4 at that time [18].

4.3 ARM Trustzone

ARM Trustzone is one well known instantiation of a TEE. It is part of a system of ARM processors integrated into System on Chips (SoCs) mostly used for smartphones. The TEE is the secure part of the processor and it runs a smaller trusted OS. It is isolated from the non-secure world, called the Rich Execution Environment, which runs the untrusted rich OS. The main hardware feature to support this split is the Non-Secure (NS) bit. The AXI bus transactions are enhanced with a NS bit so that it can block the access of secure world resources by non-secure resources. Each AXI transaction comes with this bit set or reset. When the processor runs in the secure mode, then the transaction comes with the NS bit set to zero, which gives it access to both secure and non-secure resources. When the processor runs in normal mode, it can only access resources from the normal world. This concept is extended to the level 1 and level 2 cache. These caches store an extra information bit to indicate if the code can be accessed by a secure or non-secure master. Special procedures are foreseen to jump from secure to non-secure and vice-versa. This is supported by a special monitor mode which exists in the secure world.

The split applied by ARM Trustzone is however a binary split. Applications from different vendors could co-exist together in the secure world and so if one trusted component violates the system's security, the security can no longer be guaranteed. To address this issue, protected module architectures are introduced.

Trusted Execution Environments are also being created in open-source context, more specifically in the context of the RISC-V architecture.

4.4 Protected Module Architectures and HWSW co-design solutions

If multiple software applications want to run on the same platform isolated from each other, then hardware needs to isolate them from each other at a more fine granularity. This can be done by so-called protected module architectures. The basic idea is that small software modules can run protected from all other software running on the processor. And because they are small, their properties and behavior can be verified more thoroughly. The protection is provided by extra features added to the hardware in combination with an extremely small trusted software base if needed. In the Flicker project, the software TCB relies on only 250 lines of codes but requires a dedicated TPM chip [19]. Table 12 of the review work of [18], provides an in-depth comparison of several general purpose secure processor projects with their hardware and software TCB. The hardware TCB distinguishes between the complete mother board as TCB, e.g. for TPM usage, to CPU package only for SGX and other projects. The software TCB varies from a complete secure world as is the case for TrustZone to privileged containers in the case of SGX or a trusted hypervisor, OS or security monitor.

Even more advanced are solutions with a zero trusted software base: only the hardware is trusted. This is the case for the Sancus project [20]. It implements a program counter based memory access control system. Extra hardware is provided to compare the current program counter with stored boundaries of the protected module. Access to data is only possible if the program counter is in the correct range of the code section. Progress of the program in the code section is also controlled by the hardware so that correct entry, progress and exit of the module can be guaranteed.

Intel's Software Guard Extension (SGX) are also a protection mechanism at small granularity. Software modules of an application are placed in memory enclaves. Enclaves are defined in the address space of a process, but access to enclaves is restricted. Enclaves are created, initialized, and cleared by possibly untrusted system software, but operating in the enclave can only be done by the application software. Minimizing the extra hardware to support SGX, and especially avoiding performance degradation is an important goal. The details of the hardware micro-architecture have not been disclosed: yet its most important parts are a memory encryption unit, a series of hardware enforced memory access checks and secure memory range registers [18].

4.5 Light-weight and individual solutions

The above listed solutions are mostly suited for general purpose computing, i.e. for platforms on which a complex software stack will run. In literature, more solutions are proposed to provide extremely light weight solutions to support specific security requests. SMART is one early example: it includes a small immutable piece of bootROM, considered the root of trust, to support remote attestation [21].

To protect against specific software attacks, more individual hardware countermeasures have been introduced. One example is a hardware shadow stack: to avoid buffer overflow attacks and to protect control flow integrity, return addresses are put on both the stack and the shadow stack. When a function loads a return address, the hardware will compare the return address of the stack to that of the shadow stack. They should agree for a correct return.

Another example is the protection of jump and return addresses to avoid buffer overflow attacks and other abuses of pointers. A simple but restrictive option is to use read-only memory, which fixes the pointer. A novel recent technique is the use of pointer authentication.

The authentication code relies on cryptographic primitives. A challenge for these algorithms is that they should create the authentication tag with very low latency to fit into the critical path of a microprocessor. The ARMV8-A architectures uses therefore a dedicated low-latency crypto algorithm Qarma [22]. In this approach the unused bits in a 64-bit pointer are used to store a tag. This tag is calculated based on a key and on the program state, i.e. current address and function. These tags are calculated and verified on the fly.

Address Space Layout Randomization or Stack canaries area general software technique: its aim is to make it hard to predict the destination address of the jump. A detailed description can be found in the Software Security CyBOK Knowledge Area [1].

5 HARDWARE DESIGN FOR CRYPTOGRAPHIC ALGORITHMS AT RTL LEVEL

The hardware features discussed so far are added to general purpose compute platforms, i.e. to a programmable micro-processor or micro-controller. General purpose means that a platform is created of which the hardware designer does not know the future applications that will run on it. Flexibility, reflected in the instruction set, is then of importance. A second class of processors are domain-specific processors: they have limited or no programmability and designed for one or a small class of applications.

5.1 Design process from RTL to ASIC or FPGA

When a dedicated processor is built for one or a class of cryptographic algorithms, this gives a lot of freedom to the hardware designer. Typically, the hardware designer will, starting from the cryptographic algorithm description, come up with hardware architectures at the Register Transfer Level (RTL) taking into account a set of constraints. Area is measured by gate count at RTL level. Throughput is measured by bits/sec. Power consumption is important for cooling purposes and measured in Watt. Energy, measured in Joule, is important for battery operated devices. It is often expressed in the amount of operations or amount of bits that can be processed per unit energy. Hence the design goal is to maximize the operations/Joule or bits/Joule. The resistance to side channel attacks is measured by the number of measurements or samples required to disclose the key or other sensitive material. Flexibility and programmability are difficult to measure and are typically imposed by the application or class of applications that need to be supported: will the hardware support only one or a few algorithms, encryption and/or decryption, modes of operation, initialization, requirements for key storage, and so on.

A hardware architecture is typically described in a Hardware Description Language such as Verilog of VHDL. Starting from this description the two most important hardware platforms available to a hardware designer are ASIC and FPGA. An Application Specific Integrated Circuit (ASIC) is a dedicated circuit fabricated in silicon. Once fabricated (baked) it cannot be modified anymore. A Field Programmable Gate Array (FPGA) is a special type of programmable device: it consists of regular arrays of 1-bit cells, that can programmed by means of a bitstream. This special bitstream programs each cell to a specific function, e.g. a one bit addition, a register, a multiplexer, and so on. By changing the bit-stream the functionality of the FPGA changes. From the viewpoint of the Register Transfer Level (RTL) the actual design process for either FPGA or ASIC doesn't differ that much. Similar design options are available: the designer can decide to go for serial or parallel architectures, making use of multiple design tricks to match the design with the requirements. The most well-known tricks are to use pipelining to increase throughput, or unrolling to reduce latency, time multiplexing to reduce area, etc.

From implementation viewpoint, at this register transfer abstraction level, a large body of knowledge and a large set of Electronic Design Automation (EDA) tools exist to map an application onto a FPGA or ASIC platform [3]. Implementation results should be compared not only on the number of operations, but also on memory requirements (program memory and data memory), throughput and latency requirements, energy and power requirements, bandwidth requirements and the ease with which side-channel and fault attack countermeasures can be added. Please note that this large body of knowledge exists for implementations that focus on efficiency. However, when combining efficiency with security requirements, such as constant time execution or other countermeasures, there is a huge lack of supporting EDA tools (see section 8).

5.2 Cryptographic algorithms at RTL level

Cryptographic implementations are subdivided in several categories, enumerated below. The details of the cryptographic algorithms themselves are discussed in the Cryptography CyBOK Knowledge Area [2]. Here only remarks related to the RTL implementation are made. In this section only notes specific to the hardware implementations are made.

- Secret key algorithms: both block ciphers and stream ciphers result usually in compact and fast implementations. Feistel ciphers are chosen for very area constrained designs as the encryption and decryption hardware is the same. This is e.g. not the case for the AES algorithm for which encryption and decryption require different units.
- Secret key: light-weight algorithms. For embedded devices, over the years, many light-weight algorithms have been developed and implemented, e.g. Present, Prince, Rectangle, Simon or Speck cipher. Focus in these cases is mostly on area cost. However, lately light-weight has been extended to include also low power, low energy and especially low-latency. Latency is defined as the time difference between input clear text and corresponding encrypted output or MAC. Having a short latency is important in real-time control systems, automotive, industrial IoT but also in memory encryption, control flow integrity applications etc. More knowledge will follow from the recent NIST call on light-weight crypto [23].
- Secret key: block ciphers by themselves are not directly applicable in security application. They need to be combined with modes of operation to provide confidentiality or integrity, etc. (see the Cryptography CyBOK Knowledge Area [2]). In this context efficient implementations of authenticated encryption schemes are required: this is the topic of the CAESAR competition [24]. From an implementation viewpoint, the sequential nature of the authenticated encryption schemes makes it very difficult to obtain high throughputs as pipelining cannot directly be applied.
- Hash algorithms require typically a much larger area compared to secret key algorithms. Especially the SHA3 algorithm and its different versions are large in area and slow in execution. Therefore, light-weight hash algorithms are a topic of active research.
- One important hardware application of hash functions is the mining of cryptocurrencies, such as Bitcoin, Etherium, Litecoin and others, based on SHA2, SHA256, SHA3, etc. To obtain the required high throughputs, massive parallelism and pipelining is applied. This

is however limited as hash algorithms are recursive algorithms and thus there is an upper bound on the amount of pipelining that can be applied [25]. Cryptocurrencies form part of the more general technology of distributed ledgers, which is discussed in the Distributed Systems Security CyBOK Knowledge Area [26].

- The computational complexity of public key algorithms is typically 2 or 3 orders of magnitude higher than secret key and thus its implementation 2 to 3 orders slower or larger. Especially for RSA and Elliptic curve implementations, a large body of knowledge is available, ranging from compact [27] to fast, for classic and newer curves [28].
- Algorithms resistant to attacks of quantum computers, aka post-quantum secure algorithms, are the next generation algorithms requiring implementation in existing CMOS ASIC and FPGA technology. Computational bottle-necks are the large multiplier structures, with/without the Number Theoretic Transform, the large memory requirements and the requirements on random numbers that follow specific distributions. Currently, NIST is holding a competition on post-quantum cryptography [29]. Thus it is expected that after the algorithms are decided, implementations in hardware will follow.
- Currently, the most demanding implementations for cryptographic algorithms are those used in homomorphic encryption schemes: the computational complexity, the size of the multipliers and especially the large memory requirements are the challenges to address [30].

6 SIDE-CHANNEL ATTACKS, FAULT ATTACKS AND COUNTERMEASURES

This section first provides an overview of physical attacks on implementations of cryptographic algorithms. The second part discusses a wide range of countermeasures and some open research problems. Physical attacks, mostly side-channel and fault attacks, were originally of great concern to the developers of small devices that are in the hands of attackers, especially smart-cards and pay-TV systems. The importance of these attacks and countermeasures is growing as more electronic devices are easily accessible in the context of the IoT.

6.1 Attacks

At the current state of knowledge, cryptographic algorithms have become very secure against mathematical and cryptanalytical attacks: this is certainly the case for algorithms that are standardized or that have received an extensive review in the open research literature. Currently, the weak link is mostly the implementation of algorithms in hardware and software. Information leaks from the hardware implementation through side-channel and fault attacks. A distinction is made between passive or side-channel attacks versus active or fault attacks. A second distinction can be made based on the distance of the attacker to the device: attacks can occur remotely, close to the device still non-invasive to actual invasive attacks. More details on several classes of attacks are below.

Passive Side Channel Attacks General side-channel attacks are passive observations of a compute platform. Through data dependent variations of execution time, power consumption or electro-magnetic radiation of the device, the attacker can deduce information of secret internals. Variations of execution time, power consumption or electro-magnetic radiations are typically picked up in close proximity of the device, while it is operated under normal conditions. It is important to note that the normal operation of the device is not disturbed. Thus the device is not aware that it is being attacked, which makes this attack quite powerful [31].

Side channel attacks based on variations on power consumption have been extensively studied. They are performed close to the device with access to the power supply or the power pins. One makes a distinction between Simple Power Analysis (SPA), Differential and Higher Order Power Analysis (DPA), and template attacks. In SPA, the idea is to first study the target for features that depend on the key. E.g. a typical target in timing and power attacks are if-then-else branches that are dependent on key bits. In public key algorithm implementations, such as RSA or ECC, the algorithm runs sequentially through all key bits. When the if-branch takes more or less computation time than the else-branch this can be observed from outside the chip. SPA attacks are not limited to public key algorithms, they have also been applied to secret key algorithms, or algorithms to generate prime numbers (in case they need to remain secret). So with knowledge of the internal operation of the device, SPA only requires to collect one or a few traces for analysis.

With DPA, the attacker collects multiple traces, ranging from a few tens for unprotected implementations to millions in case of protected hardware implementations. In this situation, the attacker exploits the fact that the instantaneous power consumption depends on the data that is processed. The same operation, depending on the same unknown sub-key, will result in different power consumption profiles if the data is different. The attacker will also built a statistical model of the device to estimate the power consumption as a function of the data and the different values of the subkey. Statistical analysis on these traces based on correlation analysis, mutual information and other statistical tests are applied to correlate the measured values to the statistical model.

Side channel attacks based on Electro-Magnetic radiations have been recognized early-on in the context of military communication and radio equipment. As a reaction, NATO and the governments of many countries have issued TEMPEST [32]. It consists of specifications on the protection of equipment against unintentional electro-magnetic radiation but also against leakage of information through vibrations or sound. Electro-Magnetic radiation attacks can be mounted from a distance, as explained above, but also at close proximity to the integrated circuit. Electro-Magnetic probing on top of an integrated circuit can release very localized information of specific parts of an IC by using a 2D stepper and fine electro-magnetic probers. Thus electro-magnetic evaluation has the possibility to provide more fine grained leakage information compared to power measurements.

Timing attacks are another subclass of side-channel attacks [33]. When the execution time of a cryptographic calculation or a program handling sensitive data, varies as a function of the sensitive data, then this time difference can be picked up by the attacker. A timing attack can be as simple as a key dependent different execution time of an if-branch versus an else-branch in a finite state machine. Cache attacks, which abuse the time difference between a cache hit and a cache miss are an important class of timing attacks [34], [35], .

With a template attack, the attacker will first create a copy or template of the target device [36]. This template is used to study the behavior of the device for all or a large set of inputs and secret data values. One or a few samples of the target device are then compared to the templates in the database to deduce secret information from the device. Template attacks are typically used when the original device has countermeasures against multiple executions. E.g. it might have an internal counter to log the number of failed attempts. Templates can be made based on timing, power or electro-magnetic information. As machine learning and AI techniques become more powerful, so will the attack possibility with template attacks.

Micro-architectural Side-channels Processor architectures are very vulnerable to timing attacks. The problem of information leaks and the difficulty of confinement between programs was already identified early on in [37]. Later timing variations in cache hits and misses became an important class of timing attacks [38]. Recently gaining a lot of attention are the micro-architectural side-channel attacks, such as Spectre, Meltdown, Foreshadow. They are also based on the observation of timing differences [6][38]. The strength of the attacks sits in the fact that they can be mounted remotely from software. Modern processors include multiple optimization techniques to boost performance not only with caches, but also speculative execution, out-of-order execution, branch predictors, etc. When multiple processes run on the same hardware platform, virtualization and other software techniques isolates the data of the different parties in separate memory locations. Yet, through the out-of-order execution or speculative execution (or many other variants) the hardware of the processor will access memory locations not intended for the process by means of so-called transient instructions. These instructions are executed but never committed. They have however touched memory locations, which might create side channel effects, such as variations in access time, and thus leak information.

Active fault attacks Fault attacks are active manipulations of hardware compute platforms [39]. The result is that the computation itself or the program control flow is disturbed. Faulty or no outputs are released. Even if no output is released or the device resets itself, this decision might leak sensitive information. One famous example is published in [40]: it describes an RSA signature implementation which makes use of the Chinese Remainder Theorem (CRT). With one faulty and one correct result signature, and some simple mathematical calculations, the secret signing key can be derived. Physical fault-attacks could be a simple clock glitching, power glitching, heating up or cooling down a device. These require close proximity to the device but are non-invasive.

With scaling of memories, more attack surfaces appear. A very specific attack on DRAM memories, is the RowHammer attack [41, 42]. By repeating reading specific locations in DRAM memory, neighboring locations will loose their values. Thus by hammering certain locations, bit flips will occur in nearby locations.

With more expensive equipment, and with opening the lid of the integrated circuit or etching the silicon down, even more detailed information of the circuit can be obtained. Equipment that has been used include optical fault [43], laser attacks [44], Focused Ion Beam (FIB), a Scanning Electron Microscope (SEM) and other. The latter are typically equipment that has been designed for chip reliability and failure analysis. This equipment can also be used or misused for reverse engineering.

6.2 Countermeasures

There are no generic countermeasures that resist all classes of side-channel attacks. Depending on the threat model (remote/local access, passive/active, etc.) and the assumptions made on the trusted computing base (i.e. what is and what is not included in the root of trust), countermeasures have been proposed at several levels of abstraction. The most important categories are summarized below.

To resist timing attacks, the first objective is to provide hardware that executes the application or program in constant time independent of secret inputs, keys and internal state. Depending on the time granularity of the measurement equipment of the attacker, constant time countermeasures also need to be more fine grained. At the processor architecture level, constant time means a constant number of instructions. At the RTL level, constant time means a constant number of clock cycles. At logic and circuit level, constant time means a constant logic depth or critical path independent of the input data. At instruction level, constant time can be obtained by balancing execution paths and adding dummy instructions. Sharing of resources, e.g. through caches, make constant time implementations extremely difficult to obtain.

At RTL level, we need to make sure that all instructions run in the same number of clock cycles. dummy operations or dummy gates, depending on the granularity level. Providing constant time RTL level and gate level descriptions is however a challenge as design tools, both hardware and software compilers, will for performance reasons synthesize away the dummy operations or logic which were added to balance the computations.

As many side-channel attacks rely on a large number of observations or samples, randomisation is a popular countermeasure. It is used to protect against power, electro-magnetic and timing side-channel attacks. Randomisation is a technique that can be applied at algorithm level: it is especially popular for public key algorithms, which apply techniques such as scalar blinding, or message blinding [45]. Randomisation applied at register transfer and gate level is called masking. Masking schemes randomise intermediate values in the calculations so that their power consumption can no longer be linked with the internal secrets. A large set of papers on gate level masking schemes is available, ranging from simple Boolean masking to threshold implementations that are provable secure under certain leakage models [46]. Randomisation has been effective in practice especially as a public key implementation protection measure. The protection of secret key algorithms by masking is more challenging. Some masking schemes require a huge amount of random numbers, others assume leakage models that do not always correspond to reality. In this context, novel cryptographic techniques summarized under the label leakage resilient cryptography, are developed that are inherently resistant against side-channel attacks [47, 48]. At this stage, there is still a gap between theory and practice.

Hiding is another major class of countermeasures. The idea is to reduce the signal to noise ratio by reducing the signal strength. Shielding in the context of TEMPEST is one such example. Similarly, at gate level, reducing the power signature or electro-magnetic signature of standard cells or logic modules, will increase the resistance against power or electro-magnetic attacks. Simple techniques such as using a jittery or drifting clock, and large decoupling capacitances will also reduce the signal to noise ratio.

Sometimes solutions for leaking at one abstraction level, e.g. power side channels, can be addressed at a different abstraction level. Therefore, if there is a risk that an encryption key leaks from an embedded device, a cryptographic protocol that changes the key at a sufficiently high frequency, will also avoid side-channel information leakage.

General purpose processors such as CPUs, GPUs, and micro-controllers can not be modified once fabricated. Thus protecting against micro-architectural attacks after fabrication by means of software patches and updates is extremely difficult and mostly at the cost of reduced performance [6]. Micro-code updates are also a form of software, i.e. firmware update and not a hardware update. The main difference is that the translation from instructions to micro-code is a company secret, and thus for the user it looks like a hardware update. Providing generic solutions to programmable hardware is a challenge as it is unknown beforehand which application will run. Solutions to this problem will be a combined effort between hardware and software techniques.

Protection against fault attacks are made at the register transfer level, as well as at the circuit level. At RTL, protection against fault attacks is mostly based on redundancy either in space or in time and by adding checks based on coding, such as parity checks. The price is expensive as calculations are performed multiple times. One problem with adding redundancy is that it increases the attack surface of side-channels. Indeed, due to the redundant calculations, the attacker has more traces available to perform time, power or electro-magnetic side-channel attacks [45]. At circuit level, monitors on the clock or power supply, might detect deviations from normal operations and raise an alarm.

Many type of circuit level sensors are added to integrated circuits. Examples are light sensors that detect that a lid of a package has been opened. Mesh metal sensors which are laid-out in top level metal layers can detect probing attacks. Temperature sensors detect heating or cooling of the integrated circuit. Antenna sensors to detect electro-magnetic probes close to the surface have been developed: these sensors measure a change in electro-magnetic fields. And sensors that detect manipulation of the power supply or clock can be added to the device. Note that adding sensors to detect active manipulation can again leak extra information to the side channel attacker.

Joint countermeasures against side-channel and fault attacks are challenging and an active area of research.

7 ENTROPY GENERATING BUILDING BLOCKS: RANDOM NUMBERS, PHYSICALLY UNCLONABLE FUNCTIONS

Sources of entropy are essential for security and privacy protocols. In this section two important sources of entropy related to silicon technology are discussed: random number generators and physically unclonable functions.

7.1 Random number generation

Security and privacy rely on strong cryptographic algorithms and protocols. A source of entropy is essential in these protocols: random numbers are used to generate session keys, nonces, initialization vectors, to introduce freshness, etc. Random numbers are also used to create masks in masking countermeasures, random shares in multi party computation, zero-knowledge proofs, etc. In this section the focus is on cryptographically secure random numbers as used in security applications. Random numbers are also used outside cryptography, e.g. in gaming, lottery applications, stochastic simulations, etc.

In general, random numbers are subdivided in two major classes: the Pseudo Random Number Generator (PRNG) also called Deterministic Random Bit Generator (DRBG) and the True Random Number Generator (TRNG) or Non-Deterministic Random Bit Generator (NRBG). The design, properties and testing of random numbers is described in detail by important standards, issued in the US by NIST. NIST has issued the NIST800-90A for deterministic random number generators, the NIST800-90B for entropy sources, and NIST800-90C for random bit generation constructions [49], [50] [51] [1]. In Germany and by extension in most of Europe, the German BSI has issued two important standards: the AIS-20 for functionality classes and evaluation criteria for deterministic random number generators and the AIS-31 for physical random number generators [52, 53, 54].

An ideal RNG should generate all numbers with equal probability. Secondly, these numbers should be independent from previous or next numbers generated by the RNG, called forward and backward secrecy. The probabilities are verified with statistical tests. Each standard includes a large set of statistical tests aimed at finding statistical weaknesses. Not being able to predict future values or derive previous values is important not only in many security applications, e.g. when this is used for key generation, but also in many gaming and lottery applications.

Pseudo-random number generators are deterministic algorithms that generate a sequence of bits or numbers that look random but are generated by a deterministic process. Since a PRNG is a deterministic process, when it starts with the same initial value, then the same sequence of numbers will be generated. Therefore it is essential that PRNG starts with a different start-up value each time the PRNG is initiated. This initial seed can either be generated by a slow true random number generated or at minimum by a non-repeating value, e.g. as provided by a monotonic increasing counter. A PRNG is called cryptographically secure if the attacker, who learns part of the sequence, is not able to compute any previous or future outputs. Cryptographically secure PRNGs rely on cryptographic algorithms to guarantee this forward and backward secrecy. Forward secrecy requires on top a regular reseeding to introduce new freshness into the generator. Hybrid RNG have an additional non-deterministic input to the PRNG.

PRNGs provide conditional security based on the computational complexity of the underlying cryptographic algorithms. See the Cryptography CyBOK Knowledge Area [2] for more details. In contrast, ideal true random number generators provide unconditional security as they are based on unpredictable physical phenomena. Thus their security is guaranteed independent of progress in mathematics and cryptanalysis.

The core of a true random number generator consists of an entropy source, which is a physical phenomena with a random behavior. In electronic circuits, noise or entropy sources are

[1] NIST800-90C does not exist as a standard yet.

usually based on thermal noise, jitter and metastability. These noise sources are never perfect: the bits they generate might show bias or correlation or other variations. Hence they don't have full entropy. Therefore, they are typically followed by entropy extractors or conditioners. These building blocks improve the entropy per bit of output. But as the entropy extractor are deterministic processes, they cannot increase the total entropy. So the output length will be shorter than the input length.

Due to environmental conditions, e.g. due to temperature or voltage variations, the quality of the generated numbers might vary over time. Therefore, the standards describe specific tests that should be applied at the start and continuously during the process of generating numbers. One can distinguish three main categories of tests. The first one is the total failure test, applied at the source of entropy. The second ones are online health tests to monitor the quality of the entropy extractors. The third ones are tests for the post-processed bits. The requirements for these tests are well described in the different standards and specialized text books [55].

The challenge in designing TRNGs is first to provide a clear and convincing proof of the entropy source, second the design of online tests which at the same are compact and can detect a wide range of defects [56]. The topic of attacks, countermeasures and sensors for TRNGs, especially in the context of IoT and embedded devices, is an active research topic.

7.2 Physically Unclonable Functions

From a hardware perspective, Physically Unclonable Functions (PUFs), are circuits and techniques to derive unique features from silicon circuits, similar to human biometrics [57]. The manufacturing of silicon circuits results in unique process variations which cannot be physically cloned. The basic idea of PUFs is that these unique manufacturing features are magnified and digitized so that they can be used in security applications similar to the use of fingerprints or other biometrics. Process and physical variations such as doping fluctuations, line or edge widths of interconnect wires, result in variations of threshold voltages, transistor dimensions, capacitances, etc. Thus circuits are created that are sensitive to and amplify these variations.

The major security application for PUFs is to derive unique device specific keys, e.g. for usage in an IoT device or smart card. Traditionally, this storage of device unique keys is done in non-volatile memory, as the key has to remain in the chip even when the power is turned-off. Non-volatile memory requires however extra fabrication steps, which makes chips with non-volatile memory more expense than regular standard CMOS chips. Thus PUFs are promised as cheap alternative for secure non-volatile memory, because the unique silicon fingerprint is available without the extra processing steps. Indeed, each time the key is needed, it can be read from the post-processed PUF and directly used in security protocols. They can also replace fuses, which are large and their state is relatively easy to detect under a microscope.

The second security application is to use PUFs in identification applications, e.g. for access control or tracking of goods. The input to a PUF is called a challenge, the output the response. The ideal PUF has an exponential number of unique challenge response pairs, exponential in the number of circuit elements. The uniqueness of PUFs is measured by the inter-distance between different PUFs seeing the same challenge. The ideal PUF has stable responses: it replies with the same response, i.e. there is no noise in the responses. Moreover, PUF responses should be unpredictable and physically unclonable.

The ideal PUF unfortunately does not exist. In literature, two main classes of PUFs are defined, characterized by the number of challenge-response pairs they can generate. So-called weak PUFs are circuits with a finite number of elements, with each element providing a high amount of entropy. The number of possible challenge-response pairs grows typically linear with the area of the integrated circuit. Hence they are called weak PUFs. The most well known example is the SRAM PUF [58]. These PUFs are typically used for key generation. The raw PUF output material is not directly usable for key generation as the PUF responses are affected by noise. Indeed, subsequent readings of the same PUF might result in slightly varying noisy responses, typically up to 20%. Thus after the entropy extraction follows secure sketch (similar to error correction) circuits to eliminate the noise and compress the entropy to generate a full entropy key [59]. The challenge for the PUF designer is to come up with process variations and circuits that can be used as key material, but which are not sensitive to transient noise. A second challenge is to keep all the post-processing modules compact so that the key-generation PUF can be included in embedded IoT devices.

The second class are the so-called strong PUFs. In this case, the number of challenge-response pairs grows large, ideally exponential, with the silicon area. The most well-known example is the arbiter PUF [60]. A small number of silicon elements are combined together, e.g. to create a chain of multiplexers or comparators, so that simple combinations of the elements create the large challenge-response space. Also in this case, the effects of noise in the circuits needs to be taken into account. Strong PUFs are promised to be useful in authentication applications, e.g. for access control. Each time a challenge is applied to the PUF, a response unique to the chip will be sent. The verifier will accept the response if it can be uniquely tied to the prover. This requires that the PUF responses are registered in a form of a database beforehand during an enrollment phase.

The problem with strong PUFs is that there is a strong correlation between different challenge-response pairs of most circuits proposed in literature. Hence all of these circuits are broken with machine learning techniques [61] and can not be used for authentication purposes. The fundamental problem is that very basic, mostly linear operations are used to combine PUF elements, which makes them easy targets for machine learning attacks. Ideally, these should be cryptographic or other computationally hard operations resistant to machine learning: unfortunately these cannot tolerate noise. Light-weight PUF based security protocols are an active area of research.

8 HARDWARE DESIGN PROCESS

In this section, several hardware security topics are described which are directly related to the lower design abstraction layers. One is the trust in the hardware design process itself. Directly related to this, is the problem of Trojan circuits. Also part of the hardware design process are circuit level techniques for camouflaging, logic locking, etc.

8.1 Design and fabrication of silicon integrated circuits

It is important to note that the hardware design process itself also needs to be trusted. Because of its design complexity, design at each abstraction layer relies on Electronic Design Automation (EDA) tools. The design, fabrication, packaging and test of silicon integrated circuits is an international engagement: silicon foundries are mostly located in Asia. Silicon design tools are most developed in the US, and silicon testing and packaging usually occur all over the world. For chips that end-up in critical infrastructure, such as telecommunication, military, aviation, trust and verification of the complete design cycle is essential.

Since silicon foundries and mask making are extremely expensive, very few countries and companies can still afford it and a huge consolidation has and is taking place in the industry. For critical infrastructure, governments demand more tools and techniques to increase the trustworthiness of this international design process. On this topic, large research projects are defined to come up with methods and tools to increase the trustworthiness of the design process and especially to assess the risk of Trojan insertions during the design process.

8.2 Trojan circuits

Trojan circuits are logic or gates added to large integrated circuits. As they are not part of the specified functionality, they are difficult to detect. They rely on the fact that they are extremely small in comparison with the large size of integrated circuits and SoCs. Trojan circuits are classified according to three main criteria [62, 63]. The first one is the physical characteristics of the Trojan, i.e. how is the Trojan inserted into the circuit. E.g. does it requires logic modifications or only layout modifications. The second one is the activation characteristic: will the Trojan be turned on by an internal or external event, etc. The third characteristic classifies the type of action taken by the Trojan, e.g. will it leak information or will it destroy functionality, etc. The knowledge area on this topic is summarized in [62, 63].

8.3 Circuit level techniques

To avoid visual inspection, circuit level *camouflaging* techniques are introduced [64]. These are standard cells or other modules that visually look the same, or they look camouflaged by random extra material. This is done to avoid visual inspection and reverse engineering based on visual inspection.

Another techniques to avoid loss of intellectual property is *logic locking* [65]. With this technique, extra gates are added to a circuit with a secret input. Only when the correct key is applied to the secret gates, will the circuit perform the correct functionality. This is an active research topic with logic locking schemes being proposed and attacked, with SAT solvers being a very useful tool in attacking the circuits.

8.4 Board Level Security

Integrated circuits are placed together on Printer Circuit Boards (PCBs). Many of the attacks and countermeasures mentioned before for integrated circuits, can be repeated for PCBs albeit at a different scale. While integrated circuits provide some level of protection because they are encapsulated in packages and use much smaller CMOS technologies, PCB's are less complex and somewhat easier to access. Therefore, for PCB's special coatings, and mechanical tamper evident and tamper resistant protection mechanisms could be provided. There have been some concerns that Trojan circuits could also be included at the board level.

8.5 Time

The concept of time and the concept of sequence of events are essential in security protocols. The TCG identifies three types of sequencing: a monotonic counter, a tick counter and actual trusted time [5]. A monotonic counter always increases, but the wall clock time between two increments is unknown. The tick counter increases with a set frequency. It only increases when the power is on. At power-off the tick counter will reset. Therefore the tick counter is linked with a nonce and methods are foreseen to link this with a real wall clock time. Trusted time is the most secure. It makes sure that there is a link between the tick counter and the real wall clock time. From a hardware viewpoint it will require non-volatile memory, counters, crystals, continuous power, and an on chip clock generator. The connection to a real wall clock will require synchronization and an actual communication channel.

The importance of time is placed in a wider context in the Distributed Systems Security CyBOK Knowledge Area [26].

9 CONCLUSION

Hardware security is a very broad topic, covering many different topics. In this chapter, a classification is made based on the different design abstraction layers. At each abstraction layer, the threat model, root of trust and security goals are identified.

Because of the growth of IoT, edge and cloud computing, the importance of hardware security is growing. Yet, in many cases hardware security is in conflict with other performance optimisations, such as low power or limited battery operated conditions. In these circumstances, performance optimization is *the* most important design task. Yet it is also the most important cause of information leakage. This is the case at all abstraction layers: instruction level, architecture level and logic and circuit level.

Another trend is that hardware is becoming more 'soft'. This is an important trend in processor architecture, where FPGA functionality is added to processor architectures. The fundamental assumption that hardware is immutable is lost here. This will create a whole new class of attacks.

A last big challenge for hardware security is the lack of EDA tools to support hardware security. EDA tools are made for performance optimization and security is usually an afterthought. An added challenge is that it is difficult to measure security and thus difficult to balance security versus area, throughput or power optimisations.

LINKS TO OTHER KAS

The KA on Hardware Security is linked with most other KAs in cyber security. The most important ones are mentioned in the text.

- KA in Malware & Attack Technologies
- KA in Software Security
- KA in Cryptography
- KA in Distributed Systems

REFERENCES

[1] F. Piessens, *The Cyber Security Body of Knowledge*. University of Bristol, 2021, ch. Software Security, version 1.0.1. [Online]. Available: https://www.cybok.org/

[2] N. Smart, *The Cyber Security Body of Knowledge*. University of Bristol, 2021, ch. Cryptography, version 1.0.1. [Online]. Available: https://www.cybok.org/

[3] H. Kaislin, *Top-Down Digital VLSI Design: From Architectures to Gate-Level Circuits and FPGAs*. Morgan Kaufmann, 2015.

[4] R. Anderson, *Security Engineering: a guide to building dependable distributed systems*. Wiley, 2008.

[5] D. Grawrock, *Dynamics of a Trusted Platform: A building block approach*. Intel Press, 2008.

[6] C. Canella, J. V. Bulck, M. Schwarz, M. Lipp, B. von Berg, P. Ortner, F. Piessens, D. Evtyushkin, and D. Gruss, "A systematic evaluation of transient execution attacks and defenses," *CoRR*, vol. abs/1811.05441, 2018. [Online]. Available: http://arxiv.org/abs/1811.05441

[7] National Institute of Standards and Technology, "FIPS 140-2 security requirements for cryptographic modules," may 25, 2001 (Change Notice 2, 12/3/2002). [Online]. Available: https://csrc.nist.gov/publications/detail/fips/140/2/final

[8] "NIST glossary." [Online]. Available: https://csrc.nist.gov/glossary/term/tampering

[9] "Common Criteria certified products." [Online]. Available: https://www.commoncriteriaportal.org/products/

[10] V. Lomne, "Common criteria certification of a smartcard: a technical overview," CHES 2016 Tutorial 1. [Online]. Available: https://iacr.org/workshops/ches/ches2016/presentations/CHES16-Tutorial1.pdf

[11] W. Slegers, *Security Evaluation Scheme for IoT Platforms, version 1.2*, TrustCB, 2019. [Online]. Available: https://www.trustcb.com/iot/sesip/

[12] Trusted Computing Group. [Online]. Available: https://trustedcomputinggroup.org

[13] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design - The Hardware / Software Interface (5th Edition)*, ser. The Morgan Kaufmann Series in Computer Architecture and Design. Academic Press, 2014.

[14] P. Maene, J. Goetzfried, R. D. Clercq, T. Mueller, F. Freiling, and I. Verbauwhede, "Hardware-based trusted computing architectures for isolation and attestation," *IEEE Transactions on Computers*, vol. 67, no. 3, pp. 361–374, 2017.

[15] A. P. Martin, "The ten page introduction to trusted computing," University of Oxford, Tech. Rep. CS-RR-08-11, 2008.

[16] H. Bos, *The Cyber Security Body of Knowledge*. University of Bristol, 2021, ch. Operating Systems & Virtualisation, version 1.0.1. [Online]. Available: https://www.cybok.org/

[17] Global Platform Device Committee, "EE protection profile," version 1.2, Public Release, November 2014, Document Reference: GPD_SPE_021. [Online]. Available: https://csrc.nist.gov/publications/detail/fips/140/2/final

[18] V. Costan and S. Devadas, "Intel SGX explained," Cryptology ePrint Archive, Report 2016/086, 2016, https://eprint.iacr.org/2016/086.

[19] J. M. McCune, B. J. Parno, A. Perrig, M. K. Reiter, and H. Isozaki, "Flicker: An execution infrastructure for tcb minimization," *SIGOPS Oper. Syst. Rev.*, vol. 42, no. 4, pp. 315–328, 2008.

[20] J. Noorman, J. V. Bulck, J. T. Muehlberg, F. Piessens, P. Maene, B. Preneel, I. Verbauwhede, J. Goetzfried, T. Mueller, and F. Freiling, "Sancus 2.0: A low-cost security architecture for IoT devices," *ACM Transactions on Privacy and Security*, vol. 20, no. 3, p. 33, 2017.

[21] K. Eldefrawy, G. Tsudik, A. Francillon, and D. Perito, "SMART: secure and minimal archi-

tecture for (establishing dynamic) root of trust," in *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012*, 2012.

[22] R. Avanzi, "The qarma block cipher family – almost mds matrices over rings with zero divisors, nearly symmetric even-mansour constructions with non-involutory central rounds, and search heuristics for low-latency s-boxes," Cryptology ePrint Archive, Report 2016/444, 2016, https://eprint.iacr.org/2016/444.

[23] NIST Lightweight Cryptography. [Online]. Available: https://csrc.nist.gov/projects/lightweight-cryptography

[24] CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness. [Online]. Available: https://competitions.cr.yp.to/caesar.html

[25] Y. K. Lee, H. Chan, and I. Verbauwhede, "Iteration bound analysis and throughput optimum architecture of SHA-256 (384, 512) for hardware implementations," in *Information Security Applications, 8th International Workshop, WISA 2007, Jeju Island, Korea, August 27-29, 2007, Revised Selected Papers*, 2007, pp. 102–114. [Online]. Available: https://doi.org/10.1007/978-3-540-77535-5_8

[26] N. Suri, *The Cyber Security Body of Knowledge*. University of Bristol, 2021, ch. Distributed Systems Security, version 1.0.1. [Online]. Available: https://www.cybok.org/

[27] Y. K. Lee, L. Batina, K. Sakiyama, and I. Verbauwhede, "Elliptic curve based security processor for RFID," *IEEE Transactions on Computers*, vol. 57, no. 11, pp. 1514–1527, 2008.

[28] F. Turan and I. Verbauwhede, "Compact and flexible FPGA implementation of Ed25519 and X25519," *ACM Transactions on Embedded Computing Systems*, vol. 18, no. 3, p. 21, 2019.

[29] NIST Post Quantum Cryptography. [Online]. Available: https://csrc.nist.gov/Projects/Post-Quantum-Cryptography

[30] Homomorphic Encryption Standardization. [Online]. Available: http://homomorphicencryption.org/introduction/

[31] S. Mangard, E. Oswald, and T. Popp, *Power analysis attacks - revealing the secrets of smart cards*. Springer, 2007.

[32] Link to NATO Tempest website. [Online]. Available: https://www.ia.nato.int/niapc/tempest

[33] P. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," in *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, ser. CRYPTO '96. Springer-Verlag, 1996, pp. 104–113.

[34] D. Bernstein, "Cache-timing attacks on AES," 2005. [Online]. Available: https://cr.yp.to/antiforgery/cachetiming-20050414.pdf

[35] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: The case of AES," in *Topics in Cryptology – CT-RSA 2006*. Springer Berlin Heidelberg, 2006, pp. 1–20.

[36] S. Chari, J. Rao, and P. Rohatgi, "Template attacks," in *Cryptographic Hardware and Embedded Systems - CHES 2002*. Springer Berlin Heidelberg, 2003, pp. 13–28.

[37] B. W. Lampson, "A note on the confinement problem," *Communications ACM*, vol. 16, no. 10, pp. 613–615, 1973.

[38] D. Page, "MASCAB: a Micro-Architectural Side-Channel Attack Bibliography." [Online]. Available: http://www.github.com/danpage/mascab

[39] D. Karaklajic, J.-M. Schmidt, and I. Verbauwhede, "Hardware designer's guide to fault attacks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 12, pp. 2295 – 2306, 2013.

[40] D. Boneh, R. A. DeMillo, and R. J. Lipton, "On the importance of eliminating errors in

cryptographic computations," *J. Cryptology*, vol. 14, no. 2, pp. 101–119, 2001.

[41] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors," in *Proceeding of the 41st Annual International Symposium on Computer Architecture*, ser. ISCA '14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 361–372. [Online]. Available: http://dl.acm.org/citation.cfm?id=2665671.2665726

[42] O. Mutlu, "The rowhammer problem and other issues we may face as memory becomes denser," in *Design, Automation and Test in Europe Conference, DATE 2017, Lausanne, Switzerland, March 27-31, 2017*, 2017, pp. 1116–1121.

[43] S. P. Skorobogatov and R. J. Anderson, "Optical fault induction attacks," in *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, 2002, pp. 2–12.

[44] H. Lohrke, S. Tajik, T. Krachenfels, C. Boit, and J. Seifert, "Key extraction using thermal laser stimulation: Case study on Xilinx ultrascale FPGAs," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2018, no. 3, pp. 573–595, 2018.

[45] J. Fan and I. Verbauwhede, "An updated survey on secure ECC implementations: Attacks, countermeasures and cost," in *Cryptography and Security: From Theory to Applications - Essays Dedicated to Jean-Jacques Quisquater on the Occasion of His 65th Birthday*, ser. Lecture Notes in Computer Science, D. Naccache, Ed., vol. 6805. Louvain-la-Neuve,Belgium: Springer-Verlag, 2012, pp. 265–282.

[46] S. Nikova, C. Rechberger, and V. Rijmen, "Threshold implementations against side-channel attacks and glitches," in *Information and Communications Security, 8th International Conference, ICICS 2006*, ser. Lecture Notes in Computer Science, N. Li, P. Ning, and S. Qing, Eds., vol. 4307. Raleigh, NC, USA: Springer-Verlag, 2006, pp. 529–545.

[47] D. Bernstein, "Implementing "practical leakage-resilient symmetric cryptography"," 2012, presented at CHES 2012 rumpsession. [Online]. Available: https://www.cosic.esat.kuleuven.be/ches2012/ches_rump/rs9.pdf

[48] S. Belaïd, V. Grosso, and F. Standaert, "Masking and leakage-resilient primitives: One, the other(s) or both?" *Cryptography and Communications*, vol. 7, no. 1, pp. 163–184, 2015.

[49] E. B. Barker and J. M. Kelsey, "Recommendation for random number generation using deterministic random bit generators," National Institute of Standards and Technology, Tech. Rep. Special Publication 800-90A, 2015. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf

[50] M. Turan, E. Barker, J. Kelsey, K. McKay, M. Baish, and M. Boyle, "Recommendation for the entropy sources used for random bit generation," National Institute of Standards and Technology, Tech. Rep. Special Publication 800-90B, 2018. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90B.pdf

[51] E. B. Barker and J. M. Kelsey, "Recommendation for random bit generator (RGB) constructions," National Institute of Standards and Technology, Tech. Rep. Special Publication 800-90C, Second Draft, 2016. [Online]. Available: https://csrc.nist.gov/CSRC/media/Publications/sp/800-90c/draft/documents/sp800_90c_second_draft.pdf

[52] "Functionality classes and evaluation methodology for deterministic random number generators, version 3," 2013, (In German: Funktionalitätsklassen und Evaluationsmethodologie für deterministische Zufallszahlengeneratoren). [Online]. Available: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Interpretationen/AIS_20_pdf.pdf

[53] "Functionality classes and evaluation methodology for physical random number generators, version 3," 2013, (In German: Funktionalitätsklassen und Evaluationsmethodologie für physikalische Zufallszahlengeneratoren). [Online]. Available: https://www.bsi.bund.

de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Interpretationen/AIS_31_pdf.pdf

[54] W. Killmann and W. Schindler, "A proposal for functionality classes for random number generators," 2011, aIS 20 / AIS 31, version 2.0.

[55] D. Johnston, *Random Number Generators - Principles and practices: A guide for engineers and programmers*. De Gruyter, 2018.

[56] J. Balasch, F. Bernard, V. Fischer, M. Grujic, M. Laban, O. Petura, V. Rozic, G. V. Battum, I. Verbauwhede, M. Wakker, and B. Yang, "Design and testing methodologies for true random number generators towards industry certification," in *International IEEE European Test Symposium - ETS 2018*, ser. IEEE Computer Society, Bremen,DE, 2018, p. 10.

[57] R. Maes, *Physically Unclonable Functions: Constructions, Properties and Applications*, 1st ed. Springer Publishing Company, Incorporated, 2013.

[58] J. Guajardo, S. S. Kumar, G.-J. Schrijen, and P. Tuyls, "FPGA intrinsic PUFs and their use for IP protection," in *Cryptographic Hardware and Embedded Systems - CHES 2007*. Springer Berlin Heidelberg, 2007, pp. 63–80.

[59] Y. Dodis, R. Ostrovsky, L. Reyzin, and A. Smith, "Fuzzy extractors: How to generate strong keys from biometrics and other noisy data," *SIAM J. Comput.*, vol. 38, no. 1, pp. 97–139, 2008.

[60] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas, "Silicon physical random functions," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, ser. CCS '02. ACM, 2002, pp. 148–160.

[61] J. Delvaux, R. Peeters, D. Gu, and I. Verbauwhede, "A survey on lightweight entity authentication with strong PUFs," *ACM Computing Surveys*, vol. 48, no. 2, p. 42, 2015.

[62] M. Tehranipoor and F. Koushanfar, "A survey of hardware trojan taxonomy and detection," *IEEE Design Test of Computers*, vol. 27, no. 1, pp. 10–25, 2010.

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

[64] J. Rajendran, M. Sam, O. Sinanoglu, and R. Karri, "Security analysis of integrated circuit camouflaging," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, ser. CCS '13. ACM, 2013, pp. 709–720.

[65] M. Yasin, A. Sengupta, M. T. Nabeel, M. Ashraf, J. J. Rajendran, and O. Sinanoglu, "Provably-secure logic locking: From theory to practice," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. ACM, 2017, pp. 1601–1618.

ACRONYMS

AI Artificial Intelligence.

ALU Arithmetic Logic Unit.

ASIC Application Specific Integrated Circuit.

AVA Actual Vulnerability Assessment.

CC Common Criteria.

CMOS Complementary Metal-Oxide-Semiconductor.

CPU Central Processing Unit.

CRT Chinese Remainder Theorem.

DES Data Encryption Standard.

DPA Differential and Higher Order Power Analysis.

DRAM Dynamic Random Access Memory.

DRBG Deterministic Random Bit Generator.

EAL Evaluation Assurance Level.

ECC Elliptic Curve Cryptography.

EDA Electronic Design Automation.

FIB Focused Ion Beam.

FPGA Field Programmable Gate Array.

GPU Graphics Processing Unit.

HMAC Hash MAC.

HSM Hardware Security Module.

I^{2}C Inter-Integrated Circuit.

IC Integrated Circuit.

IoT Internet of Things.

ISA Instruction Set Architecture.

ISO International Organization for Standardization.

KA Knowledge Area.

LPC Low Pin Count.

MAC Message Authentication Code.

NFC Near-Field Communication.

NIST National Institute of Standards and Technology.

NRBG Non-Deterministic Random Bit Generator.

NS Non-Secure.

OS Operating System.

PCB Printer Circuit Board.

PMA Protected Model Architecture.

PRNG Pseudo Random Number Generator.

PUF Physically Unclonable Function.

REE Rich Execution Environment.

RNG Random Number Generator.

ROM Read-Only Memory.

ROT Root of Trust.

RSA Rivest-Shamir-Adleman.

RTL Register Transfer Level.

RTM Root of Trust for Measurement.

RTR Root of Trust for Reporting.

RTS Root of Trust for Storage.

SEM Scanning Electron Microscope.

SGX Software Guard Extension.

SIM Subscriber Identification Module.

SoC System on Chip.

SPA Simple Power Analysis.

SRAM Static Random Access Memory.

TCB Trusted Computing Base.

TCG Trusted Computing Group.

TEE Trusted Execution Environment.

TPM Trusted Platform Module.

TRNG True Random Number Generator.

TXT Trusted Execution Technology.

VLSI Very Large Scale Integration.

VM Virtual Machine.

GLOSSARY

ASIC Application Specific Integrated Circuit is one class on integrated circuits, where the circuit is tuned to a specific application or set of applications. E.g. a TPM is a dedicated ASIC for security applications .

CMOS Complementary Metal Oxide Semiconductor technology is the most popular silicon technology to make integrated circuits. It consitst of complementary PMOS and NMOS transistors. Its main advantages are that it has a very low static power consumption and relative robust operation. Hence it made it possible to integrate a large number of transistors (millions to billions) into one integrated circuit.

CPU Central Processing Unit is a general purpose integrated circuit made to execute a program. It typically consists of an arithmetic unit, a program control unit, a bus structure and storage for code and data. Many types and variations exists. One SOC could contain one or more CPU cores with peripherals, extra memory, etc.

CyBOK Refers to the Cyber Security Body of Knowledge.

DRAM DRAM is Dynamic Random Access Memory. Very popular because of its high density. It requires only one transistor and one small capacitance to store one bit of data. It requires regular refreshing. It looses its value when the power supply is turned off.

FPGA A Field Programmable Gate Array or FPGA is a specialized integrated circuit that contains configurable logic, which can still be programmed after fabrication. Programming is done by loading a bitstream which configures each of the programmable logic gates individually.

GPU Graphics Processing Unit is a specialized programmable integrated circuit. Its components (arithmetic units, instruction set, memory configuration, bus structure) are all optimized to accelerate graphics, video and image processing applications.

HDL A Hardware Description Language is a special language to describe digital hardware at the register transfer level. Most well known languages are VHDL and Verilog.

IC An Integrated Circuit is an electronic device that contains a large amount of electronic components, mostly transistors integrated into one piece of semiconductor material, usually CMOS silicon. A common name is a ‘chip’ or a ‘silicon chip’ .

PCB A Printed Circuit Board is a specialized board which holds the different integrated circuits. It is made of an insulated material with copper wiring to connect the pins of different integrated circuits with each other and the outside.

RAM RAM is Random Access Memory. It is memory on an integrated circuit to store values (data or code).

Root of Trust A root of trust is a component used to realize a security function, upon which a designer relies but of which the trustworthiness can not be explicitly verified [4] .

SOC System-on-chip is a very large integrated circuit that combines multiple large components, which in previous generations might have consisted of multiple chips on one circuit board.

SRAM SRAM is Static Random Access Memory, a type of memory that makes it easy to address each individual bit, requiring typically 6 transistors per bit. SRAM looses its values when the power supply is turned off.

Trusted Computing Base The Trusted Computing Base (TCB) is the typical root of trust for a computer system. It contains all hardware and software components, that need to be trusted and of which the trustworthiness can not be explicitly verified. If security vulnerabilities occur in the TCB, then the security of the entire computer system might be at risk.

Trusted Platform Module A Trusted Platform Module is a functional component that can perform cryptographic operations, manage keys, and provide remote attestation services. When implemented as a cryptographic co-processor and embedded on a personal computer platform, it provides roots of trust so that the platform can identify itself, its current configuration, and running software..

VLSI Very Large Scale Integration is a collection of electronic design automation techniques to translate a HDL description into the actual polygons required for the maskmaking of an integrated circuit. The VLSI tools made it possible to manage the complexity of designing large integrated circuits.

INDEX