POLITECNICO DI TORINO

MASTER's Degree in ELECTRONIC ENGINEERING

MASTER's Degree Thesis

Open Hardware, Hidden Risks: Mitigating passive power side-channel leakage in RISC-V microcontrollers

Supervisors

Prof. GUIDO MASERA

PhD. MATTIA MIRIGALDI

Candidate

LORENZO CAPOBIANCO

Academic Year 2024/2025

Summary

Today's digital infrastructure is based on secure telecommunications which is crucial in all kinds of applications, from electronic payments and transaction to smart homes. In such a context, data security and confidentiality are fundamental requirements. Encryption algorithms ensure these objectives by transforming sensitive data into unintelligible information, decipherable only with the correct secret key. While these algorithms are mathematically secure, their hardware implementations remain vulnerable to side-channel attacks (SCAs), which exploit physical leakages such as timing or power consumption rather than algorithmic weaknesses. Correlations between power traces and the internal state of a device can reveal secret information, undermining cryptographic protections.

This thesis investigates the vulnerability of standard encryption schemes to SCAs, with a focus on resource-constrained embedded systems based on the royalty-free RISC-V architecture. A RISC-V microcontroller was ported to the ChipWhisperer platform, an open-source suite for hardware security evaluation, to experimentally assess leakage. The study targets two algorithms: the Advanced Encryption Standard (AES) and ASCON, a lightweight cipher explicitly designed for constrained devices with resistance against physical attacks in mind.

To deepen the analysis, the impact of substitution box (S-Box) design on side-channel resistance was examined. The original S-Boxes of AES and ASCON were compared with modified versions that enhance resistance to SCAs at the expense of certain mathematical properties, revealing a fundamental trade-off between resistance to classical cryptanalysis and physical attack resilience. The effectiveness of these design choices was validated through experimental evaluation on the RISC-V microcontroller.

Acknowledgements

I would like to thank Professor Guido Masera for giving me the opportunity to work on this thesis, whose topics kept me engaged until the very end. I am also deeply grateful to my supervisor, Mattia, for his support and guidance throughout the process, as well as to everyone at the VLSI Lab.

My sincere thanks go to all my friends for their constant encouragement during this journey, especially to my university friends, whose presence made this experience both easier and more meaningful.

Finally, a heartfelt thank you goes to my family, who have always supported me and gave me the freedom to pursue my interests.

To all of you, my deepest and most sincere gratitude.

"It always seems impossible until it's done."
Nelson Mandela

Table of Contents

List of Tables VII

List of Figures VIII

Acronyms XI

1 Introduction to Cryptography and Side-Channel Attacks 1
1.1 Side-channel attacks . . . 3

2 Background 8
2.1 Chipwhisperer . . . 8
2.2 CW305 Overview . . . 9
2.2.1 USB-to-Registers Interface . . . 11
2.2.2 Clock Domains . . . 12
2.2.3 Register Interface . . . 12
2.3 RISC-V . . . 13
2.4 OBI Protocol . . . 15
2.4.1 OBI transaction . . . 15
2.5 X-HEEP Overview . . . 17

3 Bridge 2 Xheep 20
3.1 Software model . . . 23
3.2 Hardware implementation . . . 27
3.3 Synthesis on FPGA . . . 30
3.4 On-board test . . . 31

4 Power Analysis Side Channel Attack 32
4.1 Introduction . . . 32
4.2 Simple Power Analysis (SPA) . . . 35
4.3 Differential Power Analysis (DPA) . . . 37
4.4 Correlation Power Analysis (CPA) . . . 39

4.5 Countermeasures . . . 43
4.5.1 Refactoring the algorithm . . . 43
4.5.2 Eliminate the information . . . 43
4.5.3 Suppress the side channel . . . 44

5 Advanced Encryption Standard (AES) 45
5.1 AES Robustness against Side-Channel Attacks . . . 47
5.2 Measurement Setup . . . 47
5.3 CPA Attack . . . 49
5.4 Considerations . . . 53

6 ASCON 54
6.1 Internal structure . . . 55
6.2 Sensitivity to side-channel attacks . . . 57
6.3 Measurement setup . . . 59
6.4 CPA attack . . . 60
6.5 Considerations . . . 65

7 S-Box metrics: State of the Art 67
7.1 Classical cryptanalysis resistance metrics . . . 68
7.2 Side channel attack resistance metrics . . . 69
7.2.1 AES SoA S-Boxes . . . 71
7.2.2 ASCON SoA S-Boxes . . . 71

8 Conclusion 73

Bibliography 75

List of Tables

2.1 RISC-V register definitions 14
2.2 OBI signals 15
2.3 Boot mode summary 19

3.1 Firmware file instruction endianness 26
3.2 Status Register bits function 27

5.1 Trade-off between the cryptanalytic properties and PGE threshold in AES 50
5.2 Cryptographic properties of the S-boxes in exam for AES. NL: Non-Linearity. DU : Differential Uniformity. CCV : Confusion Coefficient Variance. MCC : Minimum Confusion Coefficient. TO : Transparency Order 51

6.1 Tested S-Boxes for the analysis on ASCON with the number of collected traces 62
6.2 Trade-off between power analysis resistance and cryptanalytic properties for ASCON S-Box 62
6.3 Cryptographic properties for the S-boxes in exam for ASCON. NL: nonlinearity. DU: differential uniformity. CCV: confusion coefficient variance. MCC: minimum confusion coefficient. TO: transparency order 63
6.4 ASCON standard S-Box: attacked bits, ordered according to their SNR 63

List of Figures

1.1 It is possible to notice the peak in current absorption halfway through the transition . . . 5

2.1 CW305 target board . . . 9
2.2 CW305 hardware setup . . . 10
2.3 CW305 verilog wrapper . . . 11
2.4 Register interface read/write timings . . . 11
2.5 RISC-V logo . . . 13
2.6 The modular instruction set of the RV32IMAC variant . . . 14
2.7 OBI writing transaction . . . 16
2.8 OBI reading transaction . . . 16
2.9 X-HEEP architecture overview. . . . 17

3.1 Bridge overview . . . 20
3.2 Bridge control unit FSM . . . 21
3.3 FSM for the handshake control unit . . . 23
3.4 Bridge internal structure . . . 23
3.5 C++ software model for the *Bridge2Xheep* module. . . . 25
3.6 Firmware File - Example of not contiguous memory areas . . . 26
3.7 Software model simulation . . . 26
3.8 SystemVerilog description of the status register reset mechanism. . . 28
3.9 CW305 Simulation - USB communication . . . 29
3.10 CW305 Simulation - OBI transaction . . . 30
3.11 CW305 Simulation - Exit Boot Loop Flag . . . 30
3.12 CW305 Simulation - LED blinking . . . 31

4.1 Visualization of the short-circuit current for a logic inverter. . . . 32
4.2 Implementation of the shunt resistor as current sensor. . . . 34
4.3 SPA trace showing an entire DES operation [8]. . . . 35
4.4 Possible power traces for totally correct, partially correct and incorrect passwords. . . . 36

4.5 Python code example for the DPA algorithm. 38
4.6 Difference between aligned and unaligned power traces 38
4.7 Visual representation of the correlation factor for different relationship between the x and y variables. 40
4.8 Schematic of the measurement setup for the CPA attack 42

5.1 Visualization of the *AES* algorithm. [32] 46
5.2 Measurement setup for the CPA attack on AES 48
5.3 Power traces for the first round of AES. 50
5.4 Partial Guessing Entropy for the standard AES S-Box. 51
5.5 Partial Guessing Entropy for the S-Box Freyre 2. 51
5.6 Success rate for the standard AES S-Box. 52
5.7 Success rate for the S-Box $Freyre_2$. 52
5.8 Partial Guessing Entropy for the S-Box Freyre 1. 52
5.9 Partial Guessing Entropy for the S-Box Freyre 3. 52
5.10 Partial Guessing Entropy for the S-Box Hussain. 53
5.11 Partial Guessing Entropy for the S-Box Ozkaynak. 53
5.12 SNR for the Rijandael S-Box. 53
5.13 SNR for the Freyre 2 S-Box. 53

6.1 Ascon-AEAD128 encryption [33] 55
6.2 Display of the 3 steps of a single round [27] 57
6.3 Measurement setup for the CPA attack on ASCON 59
6.4 Plot of 40 captured power traces 60
6.5 Leakage model used for the analysis on ASCON. 61
6.6 Correlation values for the bit 33 64
6.7 Correlation values for the bit 48 64
6.8 SNR value for the bit 33 64
6.9 SNR value for the bit 48 64
6.10 Correlation plot as function of traces for bit 48 - Standard S-Box. SNR = 0.003856 65
6.11 Correlation plot as function of traces for bit 48 - Bilgin S-Box. SRN = 0.007813 65

Acronyms

SCA
Side-Channel Attack

RISC
Reduced Instruction Set Computer

ISA
Instruction Set Architecture

OBI
Open Bus Interface

X-HEEP
eXtendable Heterogeneous Energy-Efficient Platform

IP
Inellectual Property

HAL
Hardware Abstraction Layer

SDK
Software Development Kit

FMS
Finite State Machine

GPIO
General Purpose Input/Output

SPA

Simple Power Analysis

CPA

Correlation Power Analysis

DPA

Differential Power Analysis

HW

Hamming Weight

HD

Hamming Distance

S-Box

Substitution Box

AES

Advanced Encryption Standard

API

Application Programming Interface

AEAD

Authenticated Encryption with Associated Data

XOF

eXtandable Output Functions

NIST

National Institute of Standards and Technology

SNR

Signal-to-Noise ratio

NL

Non-linearity

DU

Differential Uniformity

CC

Confusion Coefficient

CCV

Confusion Coefficient Variance

MCC

Minimum Confusion Coefficient

TO

Transparency Order

VTO

improVed Transparency Order

Chapter 1

Introduction to Cryptography and Side-Channel Attacks

The security of modern e-commerce, chip-based payment cards and digital communications is ensured by data encryption and cryptographic algorithms, so that only the sender and the corresponding receiver can actually read the content of the message, preventing others from accessing this information.

The idea behind encryption is to transform ordinary information (called *plaintext*) into an unintelligible form (called *ciphertext*), by means of an encryption algorithm (*cipher*). A fundamental role in these kind of algorithms is played by the *key*, the secret value (ideally known only to the dialoguers) that allows to encrypt, and more importantly, decrypt the concealed message [1].

The need for secure communication is actually a much more ancient problem. Examples can already be identified among ancient civilisations all over the world, from the Mediterranean basin to China, via the Arab and Indian worlds. In classical times, the prime example is represented by Caesar's cipher, named after the Roman leader who invented it according to the Roman historian Suetonius [2]. In this case, the encryption algorithm consists of replacing each letter of the plaintext message with another that is three positions further down the alphabet. Although it is a very basic method of encryption, it was very effective at the time due to the very low level of education of ancient populations.

Further improvements to encryption algorithms have followed over the centuries, making use of more complex keys or more intricate encryption mechanisms, but a common aspect of all these techniques is the use of *symmetric-key cryptography*.

Symmetric-key cryptography refers to an algorithm in which both the sender

and the receiver share the same secret key, which is used to both encrypt and decrypt the message [3].

This technique, although very simple, has several drawbacks:

- The secret key has to be exchanged between the two interlocutors, so there is at least one moment when a possible malicious attacker can intercept the key.
- Once the key is obtained, the attacker is able to both decrypt any communication and generate false messages with the receiver not being aware of it.
- If the communication network consists of several nodes, it is necessary to have a key for each possible pair of interlocutors, so the number of keys to manage becomes very large and any key update results impractical.

Symmetric-key cryptography was the only known type of cryptography until the mid-1970s. In fact, in 1977 Ron Rivest, Adi Shamir and Leonard Adleman proposed an idea for a new cryptographic algorithm based on the notion of *asymmetric-key*, also known as *RSA* by the names of the three inventors.

The **asymmetric-key cryptography** [4] relies on two different but mathematically related keys, a public key and a private key, to securely transmit information. The message is encrypted using the public key, which can be distributed to anyone, and is decrypted using the private key, known only to the two communicants. The success of this implementation is linked to the way in which the two keys are generated. The RSA algorithm is in fact based on the difficulty of decomposing a very large number into two prime numbers. Therefore, even if someone has access to the encrypted information and the public key, it is very difficult for them to discover the private key needed to decrypt the message. This characteristic makes the RSA algorithm very secure, which is why it is the main type of cryptographic algorithm used in today's communications.

However, asymmetric key cryptography is inherently slow; therefore, when a large amount of data needs to be transferred, asymmetric key cryptography is only used at the beginning of the communication to exchange a common key that can be securely used for a faster and simpler symmetric key-based communication.

Although these algorithms are theoretically vulnerable to brute force attacks [5], their effectiveness is ensured by the fact that it is impossible to decrypt the message in an acceptable time if the length of the encryption key is sufficiently large, due to the computational load required.

However, other techniques, not based on mathematical analysis, can be used to extract information from encrypted data or encryption algorithms. These are commonly called *side-channel attacks*.

1.1 Side-channel attacks

For **Side Channel Attack** is intended any type of attack which has the aim of gathering extra information about the encrypted data by exploiting the way a protocol or algorithm is implemented.

It is typically assumed that the implementations of cryptographic computations are ideal "black-boxes" whose internals can neither be observed nor interfered with by any malicious entity. This assumption is necessary to enable rigorous theoretical analysis and design of cryptosystems and security protocols, but it is not realistic.

Actually, cryptographic algorithms are always implemented in software or hardware on physical devices which interact with and are influenced by their environments. These physical interactions can be instigated and monitored by adversaries and may result in information useful in cryptanalysis [6]. The effectiveness of these attacks lies in the fact that there is a correlation between the physical measurements taken at different points during the computation and the internal state of the processing device, which is itself related to the secret key. So, it is quite unlikely that an attacker would try to break the encryption algorithm by decrypting the secret key, knowing that the implementation of the algorithm has vulnerabilities that are easier to exploit.

These weaknesses can completely bypass or otherwise reduce the theoretical robustness of these algorithms.

The foundations on SCA attacks in the public cryptography research community are all due to Paul Kocher [7] [8] [9].

Side channel attacks are typically classified into three main categories [10]:

- *Control over the computation process*: it is possible to distinguish between **passive attacks** and **active attacks** on the basis of the influence carried out by the attacker on the target device. In the case of passive attacks, the attacker merely collects information on the operations of the target system while the system behaves as if there no attack occurs. In the case of active attacks, the attacker manipulates the behaviour of the target system.

- *The way the device is accessed*: it refers to the physical level of intrusiveness required by the type of attack. A distinction can be made between various cases: **invasive** attack, when it involves removing the package of a cryptographic module in order to access internal components (e.g. inserting a probe on a data bus); **semi-invasive** attack, when it is necessary to physically access the device but without damaging its passivation layer (e.g. ionising a device with a laser beam in order to modify the content of a memory); **non-invasive** attack, when the attack is limited to the mere observation of the device's operation. Attacks of the latter type exploit information unintentionally leaked by the

system. One important characteristic of non-invasive attacks is that they are completely undetectable.

- *The method used for data analysis*: SCA can be divided into **simple side-channel attack** and **differential side-channel attack**, depending on the methods used in the process of analyzing the sampled data.

Power consumption, timing analysis and electromagnetic emission are examples of *Side Channel Attacks* in digital electronic circuits, but are not the only ones. Sound, for example, is a possible method for side-channel attacks, which has been used mainly, but not only, in the past, when cryptography and communications relied on predominantly mechanical or electromechanical systems. Famous is the case of the operation *ENGULF* [11] conducted by the British MI5, which led to the decryption of secret Egyptian communications by monitoring the sound produced by the Hagelin encryption machines, or the research published in 2004 by Dmitri Asonov and Rakesh Agrawal of the IBM Almaden Research Center, which showed that the sound produced by different keys on telephones or ATM keyboards could be exploited to trace the text of the data entered [12].

For what concerns modern digital systems, side-channel attacks relies mainly on:

- Time analysis: attacks based on measuring the algorithm's computation time. By analysing the execution time of a cryptographic algorithm, it is possible to extract information about the input data. In fact, every logical operation performed by a computer takes a certain amount of time to complete and the computation time is closely related to the input data. By measuring the computation time, it is possible to trace the input data. Examples of these kind of attacks are the well known *Meltdown* and *Spectre* vulnerabilities discovered in 2018 [13] [14]. These vulnerabilities allow a process to access unauthorised memory areas by exploiting memory access time and CPUs speculative execution combined with cache side-channel attacks, thereby revealing sensitive information.

- Power analysis: attacks that exploit the variation in power consumption of the device during the execution of the cryptographic algorithm to gather information about the secret key. For example, different instructions performed by a microprocessor will have differing power consumption profiles. This kind of SCA relies on the physical properties of the electronic devices. A data bus on a PCB, for instance, consists of a metal line lying on top of a dielectric layer underneath which is a ground plane. This structure acts like a capacitor. During communication, the line is brought to logic 1 by charging the capacitor

and to logic zero by discharging it. On a physical level, commutations on the line occur through a change in the voltage at the terminals of this capacitor, which in turn is due to a shift in electrical charges. Thus, by measuring the electric current flowing on the supply line, it is possible to obtain information on the data available on the bus. This type of attack is also non-invasive, as it does not require access to the internal contents of the chip. It is sufficient to measure the current flowing through the supply line, by means of a shunt resistor.

- Electromagnetic analysis: attacks based on the measurement of the electromagnetic radiation emitted by the device, from which the secret key can be obtained directly. This type of attack has the great advantage of being totally non-invasive. Unlike power side-channel attacks, which require a shunt resistor to be inserted into the power supply, in this case, all that is needed is to place an EM probe next to the crypto processor without making any changes to the circuit. As in the previous case, power consumption $\mathbf{P}$ is measured indirectly by measuring current consumption according to the formula $p(t) = V \cdot i(t)$, since the supply voltage is constant. The key principle that enables this type of attack lies in the behaviour of logic gates during switching. In fact, the transistors used in CMOS logic pass through various conduction zones during switching, each of which corresponds to a different current level. In particular, halfway through switching, both the pull-up pMOS and the pull-down nMOS are in the active region, generating a short circuit between the power supply and ground. This phenomenon causes a peak in current consumption, known as short-circuit current, as show in Figure 1.1. Since the current flowing in a wire generates a magnetic field, this current peak in turn generates a strong variation in the magnetic field that can be measured using an EM probe. Once the power consumption has been measured in this way, the rest of the attack proceeds as for power side channel attacks.

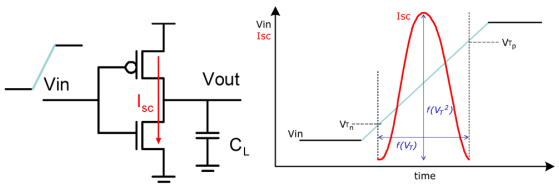


Figure 1.1: It is possible to notice the peak in current absorption halfway through the transition

Since side-channel attacks are based on the existence of correlation between the leaked information and the encrypted data, possible countermeasures mainly fall into two categories:

- Eliminate or reduce such leaked information. Possible examples of this type of countermeasure are shielding against electromagnetic emissions, so that the signal received by the attacker is weakened and the susceptibility of the chip is reduced, or physical encapsulation of the system, which reduces acoustic emissions and the possibility of probing the circuit.

- Eliminate the relation between the leaked information and the encrypted data, by using *blinding* techniques [6]. The effectiveness of side-channel attacks is due to the attacker knowing the characteristics of the encryption algorithm in the presence of some particular input. Thus, the idea behind blinding techniques is to partially alter the input data so the algorithm is brought into an unpredictable state.

Most of the countermeasures that fall into the second category are implemented at software level, since in most cases hardware manufacturers do not provide documentation on the chip's internal architecture or design choices, for commercial reasons, so that circuit-level countermeasures cannot be implemented. Unfortunately, software implementations typically have a significant timing and code size overhead as well as a substantially long development time because hands-on testing the result is crucial [15].

A further obstacle to the development of hardware countermeasures against side-channel attacks is the absence of open source tools that allow low-cost research in this area. In recent years, the rapid development and global adoption of the RISC-V platform has opened up new possibilities for security researchers.

The aim of this thesis is therefore to show that it is possible to carry out security research using completely open-source tools, such as the toolchain provided by Chipwhisperer and the RISC-V platform. The idea is to perform side-channel attacks on a RISC-V based microcontroller running encryption algorithms, measuring its power consumption in order to identify the most critical areas and develop possible hardware countermeasures.

For example, in the case of CPUs, power consumption is effectively linked to the microinstruction being executed, as this will use different elements of the chip. This phenomenon is a further source of side-channel attacks, as it allows instructions to be mapped based on their power consumption. Microarchitectural leakage can be eliminated by developing a specific instruction set (ISA) [16][17], thus working at the hardware design stage, or by adapting the code to take microarchitectural leakage into account. The second option is certainly more flexible, as it does not need physical changes to the chip, but it does require a deep understanding of the CPU's microarchitecture [18], which usually only hardware manufacturers have.

The use of a RISC-V platform allows for in-depth analysis of ISA-related microarchitectural leaks, as the analysis is set up in a complete white-box scenario where the CPU is known down to the gate-level description.

The microcontroller used for these analysis is the X-HEEP platform (eXtendable Heterogeneous Energy-Efficient Platform) [19], a 32-bit RISC-V microcontroller developed by the Embedded Systems Laboratory (ESL) at École Polytechnique Fédérale de Lausanne (EPFL), while the power measurements have been performed by using the *Chipwhisperer* framework [1], an open source toolchain dedicated to hardware security research.

The thesis is divided into 2 main parts:

– The first part is about porting the X-HEEP microcontroller to the CW305 target board developed by *Chipwhisperer*, a FPGA based evaluation board specifically designed to ease the process of side channel power analysis.

– The second part is about the actual side-channel power analysis. More specifically, an investigation is conducted into the construction of S-boxes that exploit mathematical properties aimed at reducing leakage, and their effectiveness is evaluated as lightweight countermeasures against passive side-channel attacks for the AES and ASCON algorithms.

[1] `https://github.com/newaetech/chipwhisperer`

Chapter 2

Background

2.1 Chipwhisperer

Chipwhisperer [20] is a complete open-source toolchain developed by NewAE Technology, devoted to side-channel attacks on embedded devices, with a particular focus on power analysis. This includes monitoring device's power consumption and voltage and also clock glitching attacks, which have the aim of briefly disrupting a device's power or clock to cause unintended behaviour.

The toolchain consist of three different layers:

- **Hardware**: Chipwhisperer provides boards specifically developed for SCA. This includes *scope* boards, which are used to mount the side-channel attacks, and *target* boards, which works as a device under test (DUT). As far as *scope* boards are concerned, they can measure power consumption, allowing an encryption key to be retrieved, or voltage glitches to wreak havoc on an embedded device. *Target* boards include all the necessary tools (e.g. shunt resistor on the supply line etc) to test the resistance of the developed hardware against side channel attacks. Among these there is the CW305 board, which is analysed in more detail in the Section 2.2.

- **Firmware**: Chipwhisperer provides firmware for all their evaluation boards, which means C code for the MCU used as the USB controller and Verilog HDL for FPGA applications. An example is the Verilog wrapper used to facilitate the insertion of a proprietary core on the CW305 board's FPGA.

- **Software**: Chipwhisperer software includes a Python API for talking to ChipWhisperer hardware (ChipWhisperer Capture), via the USB controller, and a Python API for processing power traces from ChipWhisperer hardware (ChipWhisperer Analyzer).

One unique strength of Chipwhisperer scope boards is that the sampling clock used for the power consumption measurements is derived directly from the device clock, ensuring a degree of synchronism on measurements that would be impossible to achieve with an external oscilloscope. With a real-time oscilloscope, in fact, the internal sample clock of the oscilloscope will be running at all times, and the sample occurs at the next clock edge after the trigger. Thus even though the oscilloscope is triggered at a repeatable time, there will be some random jitter between when the first sample occurs relative to this trigger for unsynchronized (free-running) sample clocks. This approach relaxes the sample rate requirements.

2.2 CW305 Overview

The CW305 is a FPGA target board developed by *Chipwhisperer* to ease the process of power side channel attack analysis. The board consist of an Artix A100 FPGA, that implements a template structure in which it is easy to insert an IP design, and a SAM3U microcontroller, which manages the USB communication with the host PC. An overview of the board is shown in Figure 2.1:

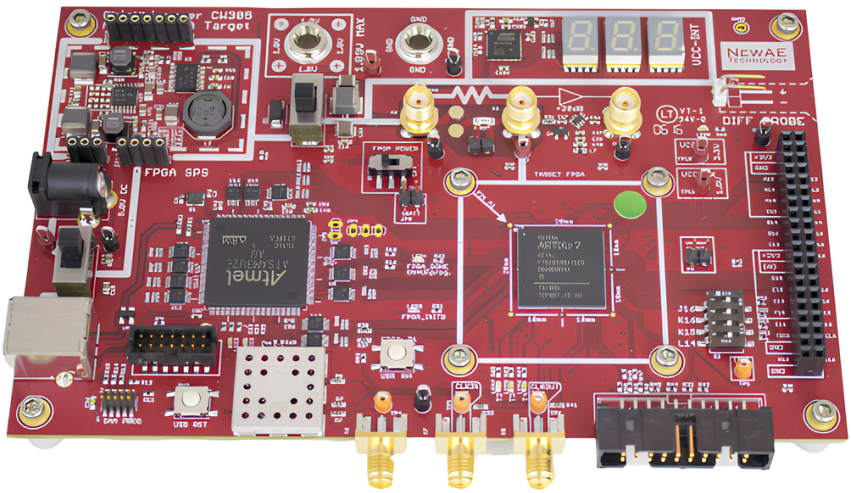


Figure 2.1: CW305 target board

The custom USB interface, managed by the SAM3U microcontroller, allows data to be sent and received from the FPGA directly from the host PC used to process the measurement data. At the physical level, the CW305 provides an address/data bus between the microcontroller with USB interface and the FPGA. This address/data bus allows arbitrary data to be read/written from/to the FPGA by simply selecting the appropriate register address. This solution also allows

the board's configuration parameters, such as the PLL output frequency, to be adjusted via the Python API to meet design requirements. Some parameters, such as the clock source or the FPGA bitstream loading method, can also be set via the on-board DIP switches. The measurement set-up schematic is shown in the Figure 2.2. The capture board in the schematic can also be replaced by an external oscilloscope.

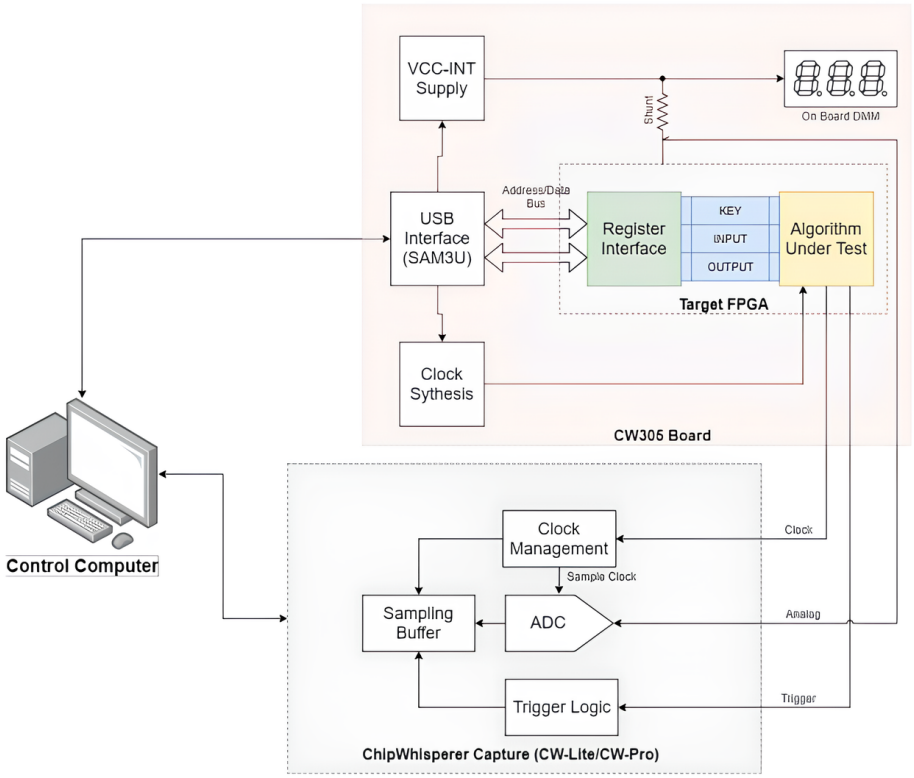


Figure 2.2: CW305 hardware setup

The implementation of proprietary cores on the FPGA is facilitated by the presence of a Verilog wrapper, whose purpose is to decouple the design of the core module from the hardware infrastructure needed to communicate with the control computer.

The top-level **cw305_top.v** wrapper consists of 3 Verilog sub-modules, as shown in Figure 2.3:

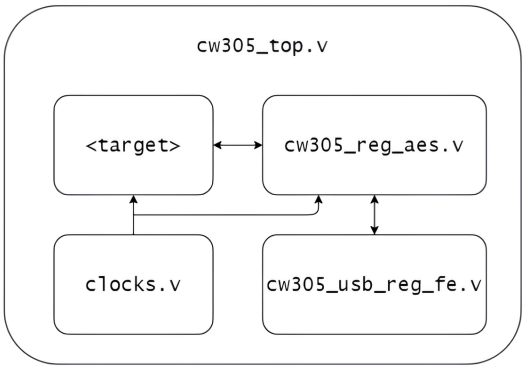


Figure 2.3: CW305 verilog wrapper

- **cw305_usb_reg_fe.v**: Front-end to the SAM3U USB interface. This module contains all the logic needed to correctly interface the register block on the FPGA with the SAM3U microcontroller, which manages the USB communication with the PC.
- **clocks.v**: controls the routing of the internal and external clock signals.
- **cw305_reg_crypto.v**: register block. This module includes all the control and status registers that the ChipWhisperer software interacts with in order to control the target. For instance, Figure 2.3 shows the term *cw305_reg_aes.v* since a AES cryptographic core is instantiated.

2.2.1 USB-to-Registers Interface

This module (*cw305_usb_reg_fe.v*) manages the USB communication between the SAM3U microcontroller and the FPGA registers block. It defines the timing requirements for the registers read and write operations, as shown in Figure 2.4:

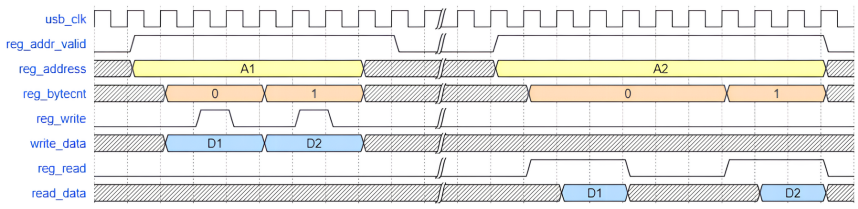


Figure 2.4: Register interface read/write timings

The interface between the microcontroller and the USB-to-Registers module is based on an 8-bit data bus and a 21-bit address bus. The first 14 MSBs of the

address are used to address a physical register, while the remaining 7 bits are used to address the byte within the same register. In this way, each register can store up to 128 bytes.

The read and write operations are managed by 3 control signals in the CW305 top level:

- CE#: active low chip enable
- RD#: active low read signal
- WR#: active low write signal

At the positive edge of the *usb_clk* signal, if CE = 0 and WR = 0 the data on the *usb_data* bus is sampled and stored in the corresponding register pointed by the *usb_addr* value.

On the contrary, if CE = 0 and RD = 0 at the positive edge of the *usb_clk* signal, the data on the *usb_data* bus is available and can be sampled by the host PC.

2.2.2 Clock Domains

The CW305 FPGA board supports 2 different clock domains: the USB clock domain, used for all the control and status register the ChipWhisperer software interacts with directly, and the cryptography clock domain, for the target logic.

This structure has 2 main benefits: first of all, the USB clock can be disabled when performing power measurements, ensuring that the captured traces are not affected by USB-clock related noise; secondly, the USB clock is fixed at the frequency of 96 MHz, while the cryptography clock can be changed according to the target needs and the on-board PLL specifications.

Clocks routing is defined in the *clocks.v* module.

2.2.3 Register Interface

This module (*cw305_reg_crypto.v*) interfaces with both the target module and the USB-to-register module. It contains all the control and status registers used by the ChipWhisperer software to interact with the board, allowing the direct control of the target module from the external PC.

Register address definitions are located in a separate file, called cw305_defines.v. This allows to easily change the names and the behaviour of the registers according to the design needs. When connecting to the target board using the Python APIs, this Verilog defines file is automatically parsed so that registers may be referred to

by their Verilog name. The contents of the registers can be passed to the device under test according to the project specifications.

2.3 RISC-V

Since power consumption measurements are performed on a RISC-V-based microcontroller, it is worth analysing this architecture in a deeper level of detail.

RISC-V is an open standard *instruction set architecture* (ISA) based on the principle of *Reduced Instruction Set Computer* (RISC). The project began in May 2010 at the University of California, Berkeley, when prof. Krste Asanović and graduate students Yunsup Lee and Andrew Waterman [21] started it as part of the Parallel Computing Laboratory (Par Lab) at UC Berkeley, of which Prof. David Patterson was Director. The latter in particular was a pioneer of the RISC architecture and one of the first to argue the obvious benefits of this approach over its CISC counterpart, including a much simplified control system and faster execution speed of individual instructions, which overall resulted in higher CPU performance. The name RISC-V was actually a way of honouring Berkeley's earlier RISC projects (RISC-I, RISC-II, ...) that started to appear in the 1980s thanks to Professor Patterson himself.

Figure 2.5: RISC-V logo

The idea behind the project was to create a modular ISA [22] (a set of instructions executable by a microprocessor) that would be royalty-free and usable by anyone, both academically and commercially.

Modularity is a crucial aspect to ensure adoption of the instruction set by as many entities as possible, from simple academic projects to complex industrial designs. For this reason, the ISA is composed of instruction subsets that allow the hardware to be developed according to the project's needs. The base alone (shown in Figure 2.6) can implement a simplified general-purpose computer, with full software support, including a general-purpose compiler. Being a RISC based architecture, memory accesses are reserved to load and store instructions only.

The standard supports a 32 x 32-bits register file and also defines the specific function of each register, as show in Table 2.1. When the floating-point extension is implemented, an additional 32 floating-point registers are placed. Except for memory access instructions, instructions address only registers. The first integer register is a zero register, and the remainder are general-purpose registers. A store

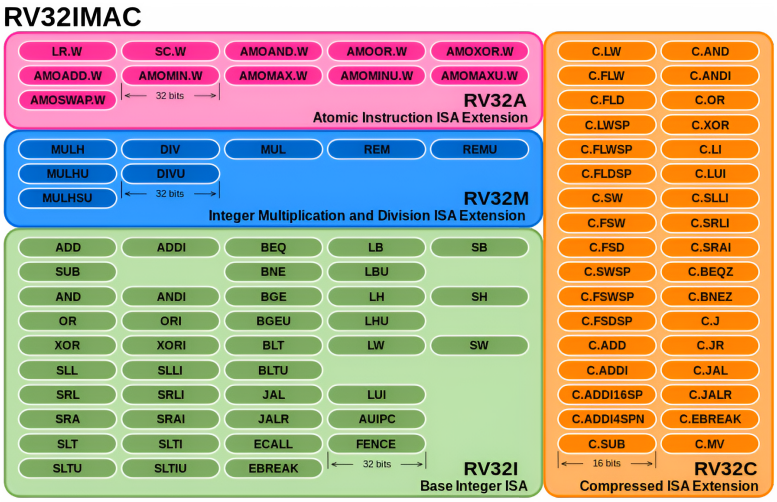


Figure 2.6: The modular instruction set of the RV32IMAC variant

to the zero register has no effect, and a read always provides 0. Using the zero register as a placeholder makes for a simpler instruction set.

Register Name	Symbolic Name	Description	Saved by
32 integer registers			
x0	zero	Always zero	
x1	ra	Return Address	Caller
x2	sp	Stack Pointer	Callee
x3	gp	Global Pointer	
x4	tp	Thread Pointer	
x5	t0	Temporary / alternate return address	Caller
x6-7	t1–2	Temporaries	Caller
x8	s0/fp	Saved register / frame pointer	Callee
x9	s1	Saved register	Callee
x10–11	a0–1	Function arguments / return values	Caller
x12–17	a2–7	Function arguments	Caller
x18–27	s2–11	Saved registers	Callee
x28–31	t3–6	Temporaries	Caller

Table 2.1: RISC-V register definitions

2.4 OBI Protocol

The *Open Bus Interface* (OBI) is a request-grant-based protocol similar to the AMBA AXI protocol. It was proposed by the OpenHW Group as the standard communication bus for their *CV32E40P* processor [1] — a 4-stage, in-order, 32-bit RISC-V core — which is one of the CPUs available for the X-HEEP platform. Although it is a generic communication protocol, the processor mainly uses it for interfacing with code and data memories. Within X-HEEP, it is also used as a communication protocol for the extension bus, which is key to this platform's flexibility.

The OBI is easier to interface than the AMBA AXI and AHB protocols, and improves timing by removing rvalid->req dependency. The protocol also forces address stability. Thus, the core cannot retract memory requests once issued, nor can it change the issued address. Communication relies on the main signals reported in Table 2.2:

Name	Source	Destination	Description
Global signals			
clk	Clock source	All	The bus clock times all bus transfers. All signal timings are related to the rising edge of clk.
reset_n	Reset controller	All	The bus reset signal is active LOW and resets the system and the bus. This is the only active LOW signal.
Address channel (A) signals			
req	Manager	Subordinate	Address transfer request. req=1 signals the availability of valid address phase signals.
we	Manager	Subordinate	Write Enable, high for writes, low for reads.
be[]	Manager	Subordinate	Byte Enable. Is set for the bytes to write/read.
addr[]	Manager	Subordinate	Address
wdata[]	Manager	Subordinate	Write data. Only valid for write transactions. Undefined for read transactions.
rready[]	Manager	Subordinate	Ready to accept response transfer. Response transfer is accepted on rising clk with rvalid=1 and rready=1.
Response channel (R) signals			
gnt	Subordinate	Manager	Grant. Ready to accept address transfer. Address transfer is accepted on rising clk with req=1 and gnt=1.
rvalid	Subordinate	Manager	Response transfer request. rvalid=1 signals the availability of valid response phase signals. Used for both reads and writes.
rdata[]	Subordinate	Manager	Read data. Only valid for read transactions. Undefined for write transactions.

Table 2.2: OBI signals

2.4.1 OBI transaction

Each OBI transaction consists of two transfers:

1. **Address phase transfer** over the **A channel**.
2. **Response phase transfer** over the **R channel**.

[1] `https://docs.openhwgroup.org/projects/cv32e40p-user-manual/en/latest/intro.html`

1. Address Phase Transfer

1. The manager asserts the request signal (`req`) high to indicate that the address phase signals (`addr`, `wdata`, `we`, `be`) are valid.
2. The subordinate asserts the grant signal (`gnt`) high to indicate it is ready to accept the address phase signals.
3. The address phase of a transaction **starts** in the cycle where `req` goes high, and **completes** on the rising edge of the clock when both `req` and `gnt` are high.

2. Response Phase Transfer

1. After a granted request on the A channel, the subordinate asserts `rvalid` high to indicate that the response signals (e.g., `rdata`) are valid.
2. The manager asserts `rready` high to indicate it is ready to accept the response phase signals.
3. The response phase of a transaction **starts** in the cycle where `rvalid` goes high, and **completes** on the rising clock edge when both `rvalid` and `rready` are high.

The OBI protocol supports other additional signals (e.g. `auser`, `wuser`, `aid`, etc.), the usage of which is specified in the reference manual, but these are extensions of the basic protocol and are therefore not supported by all controllers that comply with the OBI standard. The basic OBI transfers are illustrated in the Figure 2.7 and Figure 2.8.

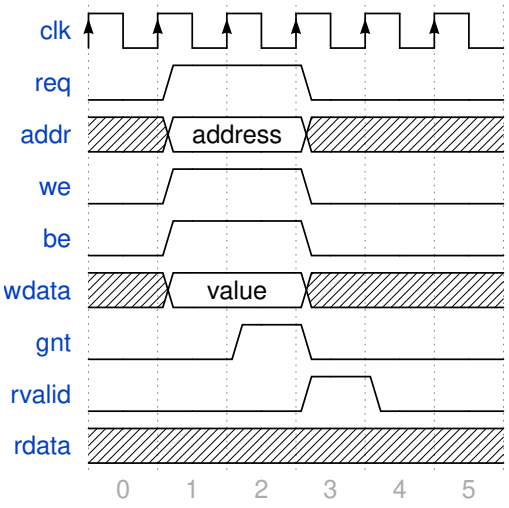


Figure 2.7: OBI writing transaction

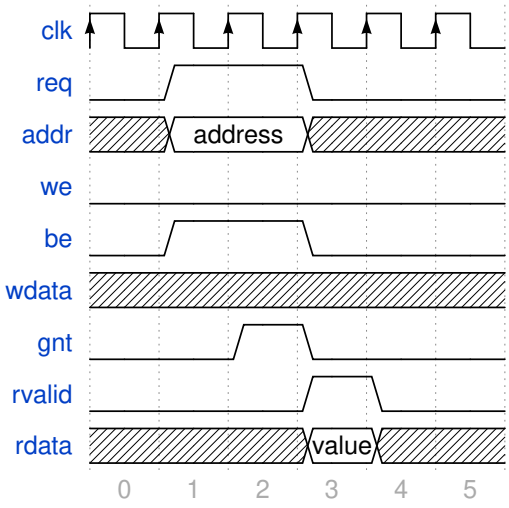


Figure 2.8: OBI reading transaction

2.5 X-HEEP Overview

X-HEEP [19] is an open-source, extensible and configurable single-core RISC-V microcontroller developed at the Embedded Systems Laboratory(ESL) of EPFL, designed to be used both as low-cost microcontroller or extended and customized with external peripherals and accelerators. Integration with external peripherals is guaranteed by a shared bus system, which exposes master and slave ports to/from the bus. The communication between the CPU and the external peripherals relies on the OBI protocol described in the previous section.

This approach allows to inherit an IP fully capable of booting RTOS (such as FreeRTOS) with the whole firmware stack, including HAL drivers and SDK, but modular and extensible with proprietary hardware co-processors, accelerators or any type of IP. An overview of the complete architecture is shown in Figure 2.9.

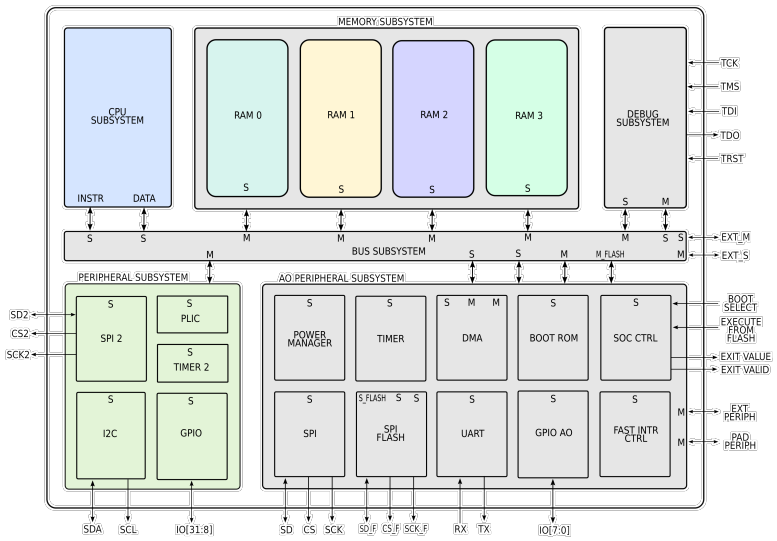


Figure 2.9: X-HEEP architecture overview.

The whole system is divided into 4 different power domains: CPU subsystem domain, memory banks domains, peripheral subsystem domain and always-on peripheral subsystem domain. IPs are carefully selected and grouped in the mentioned power domains in order to maximize energy savings for ultra-low-power edge-computing applications. A noteworthy aspect is that many of the IPs within the design are actually derived from existing major open-source projects, such as *PULP*, *OpenHW* and *lowRISC*.

The **CPU subsystem** is itself modular. The CPU can be selected among some of the RISC-V-based processors developed by the OpenHW group, such as CVE2,

CV32E40P and its variant with the CORE-V-X Interface called CV32E40PX, and the CV32E40X, also with the X-IF. All the CPUs are 32-bit, open-source, low-power embedded cores featuring a Harvard architecture with two separate bus interfaces for instructions and data. Both interfaces use the Open Bus Interface (OBI) protocol. Since the whole system has been designed to be energy-efficient, this domain supports clock gating in order to reduce power dissipation when the CPU is not needed.

The **memory subsystem** is divided into multiple banks, used for both code and data memories. Each bank can have a different size. Access to multiple banks simultaneously is possible thanks to dedicated interfaces with the global bus system. Each bank supports both clock gating and retention modes to save power.

The **peripheral subsystem domain** includes some peripherals which are not indispensable to the operation of the microcontroller but are convenient to have in real-world applications, such as a general-purpose timer, an interrupt controller, an I2C interface, a serial peripheral interface (SPI) and 24 general-purpose input-output (GPIO). This subsystem is meant to be clock-gated or switched off when not used.

The **always-on peripheral subsystem domain** consists of all those peripherals designed to be powered on all the time. For this reason, it is not possible to apply power-saving techniques to this domain. These include the power manager, the fast interrupt controller and the boot ROM.

In particular, the *boot ROM* is crucial for system operation, as it determines where the microcontroller jumps at reset time. The boot ROM contains code for three different booting modes, which can be used for simulation and synthesis on an FPGA.:

- JTAG
- SPI Flash Execution
- SPI Flash Loading

The boot procedure is defined by the value of two specific signals, `boot_sel_i` and `execute_from_flash_i`, as reported in Table 2.3.

`boot_sel_i`	`execute_from_flash_i`	Boot procedure
0	X	JTAG
1	1	SPI Flash Execution
1	0	SPI Flash Loading

Table 2.3: Boot mode summary

JTAG When this boot mode is selected, the CPU enters the boot ROM and loops indefinitely until an external JTAG programmer sets a specific exit loop flag to 1 by writing to the corresponding memory location. During this process, the CPU runs in debug mode, waiting for the programmer to finish loading instructions into RAM before starting program execution. Subsequently, the CPU begins executing instructions from the specified start address, which is obtained from a memory-mapped register and is set to `0x180` at reset time.

An external bridge was added using the X-HEEP extensible structure to communicate with both the CW305 register domain and the microcontroller itself in order to load firmware into RAM when this boot mode is selected.

SPI Flash Execution During this boot procedure, the CPU jumps to the FLASH memory to execute the code directly from it once it enters the boot ROM. Memory access operations are automatically translated into SPI transactions. This method is mainly used as a second-stage boot procedure since it is very slow.

SPI Flash Loading When this boot mode is selected, the CPU enters the boot ROM and, via the OpenTitan SPI host, copies the first 1 kB of content from the FLASH (starting at address 0) to the RAM (also starting at address 0). The CPU then jumps to the entry point at `0x180` in RAM and executes the start function of the `crt0` file, which is contained within the 1 kB of code copied to RAM. This function checks whether the code has been completely copied (i.e. whether it is less than or equal to 1 KB). If so, it jumps to the main function. Otherwise, it uses the OpenTitan SPI to copy the remaining code bytes.

Chapter 3

Bridge 2 Xheep

During the boot phase, the *X-HEEP* microcontroller requires instructions to be loaded into RAM from either an external flash memory or a JTAG programmer. To allow the firmware to be uploaded via the structure and API provided by *Chipwhisperer* instead, a module called `Bridge2Xheep` was developed. The `Bridge2Xheep` module interfaces the *X-HEEP* microcontroller and the registers block. This allows data sent from the host PC via USB to be loaded into the microcontroller's memory, eliminating the need for an external JTAG programmer (e.g. GDB).

The bridge structure is shown in Figure 3.1.

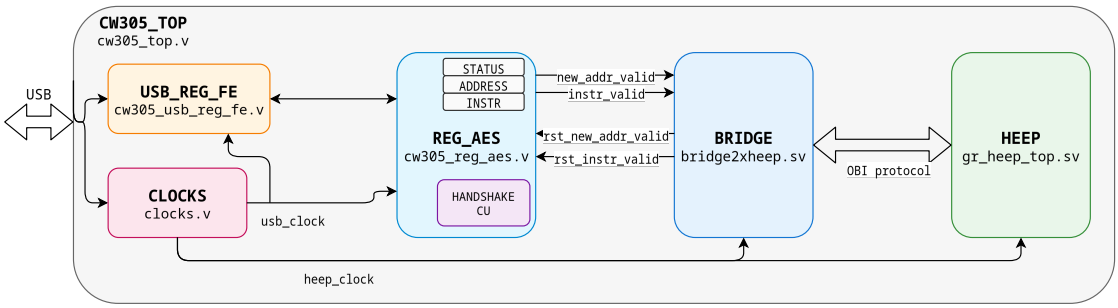


Figure 3.1: Bridge overview

The firmware upload procedure consists of several steps. The host PC reads the firmware file, extracts the instructions in the correct format, and sends the data to the CW305 register interface via the Python API one byte at a time. Once the data has been sent to the correct board register, the PC sets one of the status flags in the status register to 1, thereby signalling to the bridge that a new instruction or address is valid. The host PC then waits for the status flags to return to 0 before starting a new operation. Meanwhile, the bridge continuously polls the value of the status register bits until a new address or instruction is detected. When this occurs, the bridge initiates communication with *X-HEEP* via the *OBI* protocol and

waits for the microcontroller to assert the grant signal. Once this has happened, the microcontroller has received the data correctly and communication can be considered concluded. The bridge will then reset the status register flags, allowing the host PC to send new data.

A specific control unit was designed within the bridge to support this protocol, as shown in Figure 3.2.

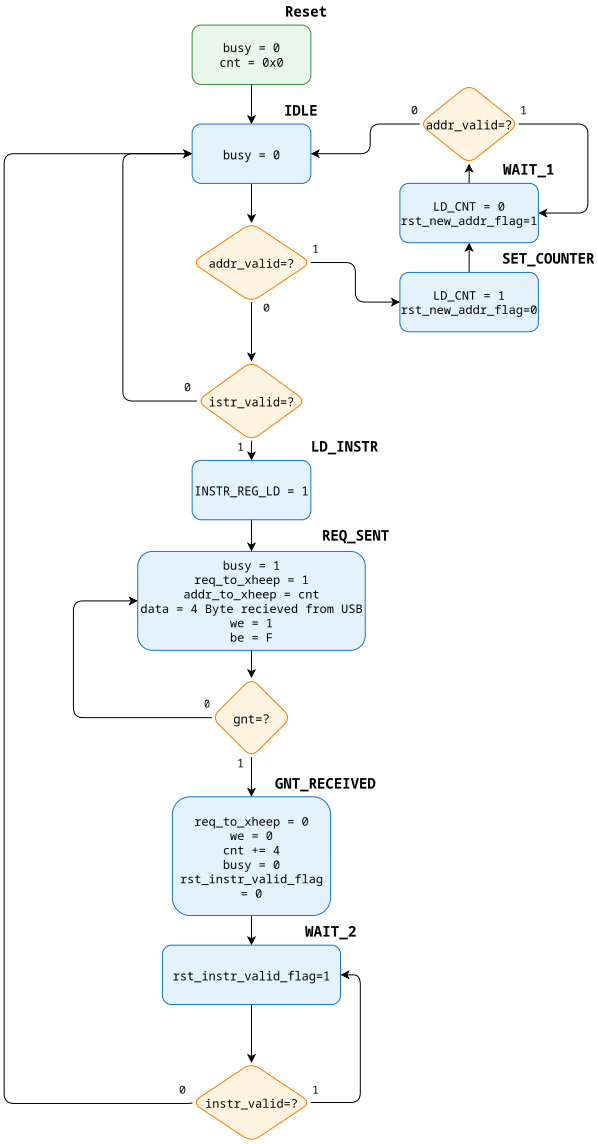


Figure 3.2: Bridge control unit FSM

More in detail, the behaviour of the bridge is different in the case of a valid address or a valid instruction. In fact, when a new address is available, the bridge

merely updates the value of its internal counter with the new address provided by the host PC, while, in the latter case the communication with *X-HEEP* is started. The bridge also supports a *busy* flag, which signals the bridge availability, although it is not actually used in the design.

Since the bridge module acts as an intermediary between the register block and the microcontroller, the communication protocol differs depending on which side the bridge is communicating with. On the *X-HEEP* side, the communication relies on the OBI protocol explained in the Section 2.4. It is important to notice that the bridge is connected to the *X-HEEP* bus master port (`EXT_M` in Figure 2.9). On the register side, a simple handshake protocol has been implemented to prevent errors when the bridge and register block need to communicate, as they operate in different clock domains. As explained in Section 2.2.2 and shown in Figure 3.1, the entire design consists of two different clock domains: the USB clock and the *X-HEEP* clock. The former operates at a fixed frequency of 96 MHz, while the latter's frequency can be adjusted according to the target device's synthesis timing requirements.

A special control unit manages the handshake protocol for the USB clock domain, while the *X-HEEP* clock domain protocol counterpart is managed by the bridge.

A schematic of the USB-side handshake control unit is shown in Figure 3.3. The main purpose of this module is to translate the reset signals received from the bridge into pulses with a duration equal to one clock cycle of the USB clock domain, as the two blocks operate at different speeds.

The bridge internal structure is shown in Figure 3.4 and it consists of:

- A counter, used for address management in OBI transactions. The counter allows both increasing the internal value and loading a new address. This is necessary because the firmware file is not contiguous and sometimes it is necessary to jump to new memory areas. As the instructions consist of 32 bits, the counter increases by four units each time a new instruction is sent correctly to the microcontroller.
- A 32-bit register, used to temporary store the instruction
- A control unit, which manages the communications with both *X-HEEP* and the registers domain.

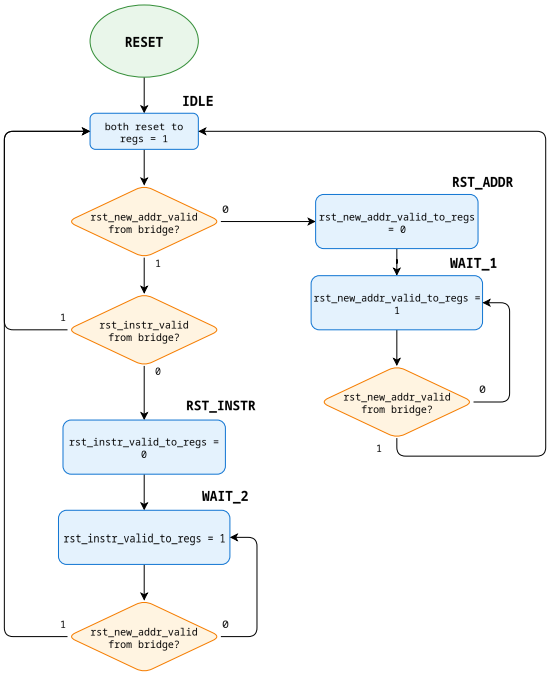


Figure 3.3: FSM for the handshake control unit

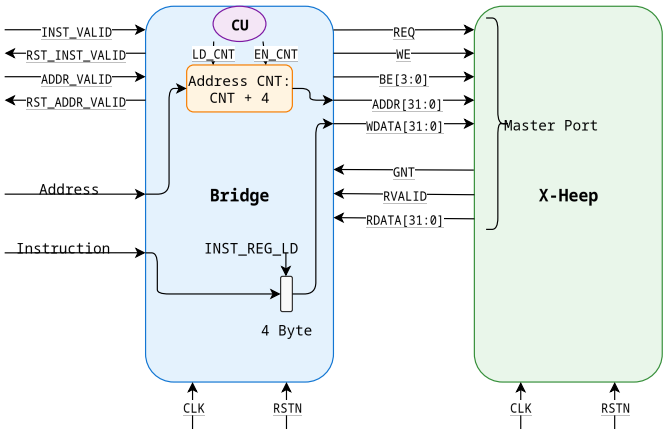


Figure 3.4: Bridge internal structure

3.1 Software model

A first software model was implemented to ensure the correct interfacing with *X-HEEP* only. The model has been implemented as a C++ class, and then tested using Verilator [23], a software programming tool which converts the hardware description language Verilog to a cycle-accurate behavioural model in the programming languages C++ or SystemC. The bridge software model consists of a *Driver*

class, which simulates the OBI communication with the microcontroller, as shown in Listing 3.5:

```
void Drv::drive(ReqBridge *req)
{
    if (req->valid)
    {
        if (req->address >= 0x180)
        {
            switch (this->state)
            {
            case 0:
                // IDLE state

                dut->req_i = 0;
                this->state = 1;
                this->busy = 1;
                break;

            case 1:
                // REQUEST_SENT state

                dut->req_i = 1;
                dut->addr_i = req->address;
                dut->wdata_i = req->instruction;
                dut->we_i = 1;
                dut->be_i = 0b1111;

                if (dut->gnt_o)
                {
                    this->state = 2;
                    this->busy = 0;
                } else {
                    this->state = 1;
                }

                break;

            case 2:
                // GNT RECEIVED

                dut->req_i = 0;
                dut->we_i = 0;
```

```
41
42                  req->address += 4;
43                  req->valid = 0;
44
45                  this->state = 0;
46                  break;
47
48              default:
49                  this->state = 0;
50                  this->busy = 0;
51                  break;
52              }
53          }
54
55      }
56
57  }
```

Figure 3.5: C++ software model for the *Bridge2Xheep* module.

This model is a C++ translation of the FSM shown in Figure 3.2: a *state* variable is used to store information about the current state, while a switch case statement determines the state evolution and the active signals for each state.

It can be noticed that this model does not foresee any handshake mechanism towards the register interface, as integration with the structure provided by the CW305 board is not yet considered at this stage of the project. In addition, a *Request* class was used to temporarily store the address and instruction to be sent to the microcontroller.

The model has been simulated using a C++ testbench, structured as follows:

- Firstly, the Device-under-Test is reset and initialised
- After that, a special function has the task of reading the firmware file and extracting the instructions with the correct endianness. When new data is available, a new request is sent to X-HEEP. The request is kept active until the model receives a response from the microcontroller. This procedure goes on until all firmware has been loaded into RAM.
- When the loading procedure is complete, the boot exit loop flag is set by the `tb_set_exit_loop` method.

This first simulation revealed an important detail about the firmware file format. The endianness of the instructions in the file—automatically generated by `gcc`—is

opposite to what is expected by the microcontroller. As a result, it was necessary to rearrange the characters in groups of two to match the required format. For clarity, an example is provided in Table 3.1:

Firmware file instruction format	Expected instruction format
97 F1 00 00	00 00 F1 97

Table 3.1: Firmware file instruction endianness

Furthermore, the firmware file is not contiguous. The instructions in the file only refer to non-zero memory areas, which are marked by an address that defines their starting point. When a new address is encountered, the previous block has ended, meaning that all the remaining instructions between the two addresses are to be considered 0. This becomes problematic when the last instruction in a memory area contains trailing zeros, as these are omitted. Consequently, the instruction will be expressed over fewer than 4 bytes, and this must be taken into account in the reading algorithm to maintain address/data alignment. As can be seen in Figure 3.6, on line 26, the last instruction is reported as `82 80`, when it should be `82 80 00 00`.

```
25  C0 1B EF 10 80 1F 02 45 4C 00 01 46 EF 00 20 04
26  6F 00 60 01 EF 00 30 68 82 80
27  @000001EC
28  41 11 7D 55 06 C6 EF 00 D0 67 41 11 89 67 22 C4
29  06 C6 93 87 C7 85 2A 84 81 C7 81 45 EF 10 40 65
30  BD 67 03 A5 C7 E3 1C 55 91 C3 82 97 22 85 EF 00
```

Figure 3.6: Firmware File - Example of not contiguous memory areas

Taking these details into account, the model proved to be functional and the communication with *X-HEEP* was correct. A screenshot of the *Verilator* simulation is shown in Figure 3.7.

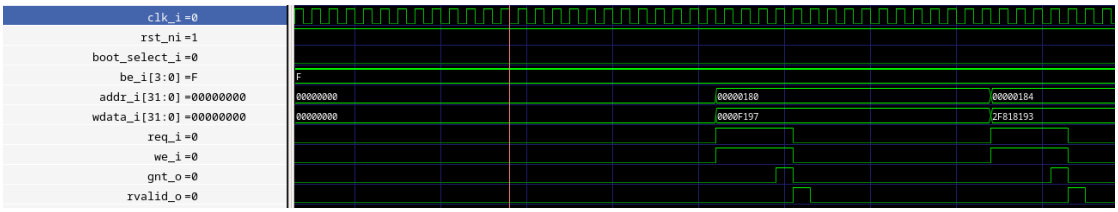


Figure 3.7: Software model simulation

3.2 Hardware implementation

After verifying through simulation that the software model was successful, a hardware model of the bridge was implemented. The model was described in SystemVerilog to remain consistent with the *X-HEEP* project, which is also described in this language.

The hardware model follows the same structure shown in Figure 3.4 and the relative control unit implements the same algorithm shown in Figure 3.2. In particular, the microcontroller interface provides all the signals required for the OBI protocol, and the registers interface also supports additional ports such as the busy signal and a dedicated bus for data received from the microcontroller, even though these are not used in the design.

At this stage of the project, it became clear that a synchronisation mechanism was required to enable data exchange between the bridge and the registers, which are controlled directly by the Python API. The solution was to use some bits of a special register, namely the *Status Register*, as flags to signal data availability.

The *Status Register* has 8-bit parallelism (1 byte), which is used as follows:

Status Register Bit	Function
Status[0]	Bridge Busy Flag
Status[1]	Instruction Valid
Status[2]	Address Valid
Status[3]	Trigger Program Execution
Status[4]	Unused
Status[5]	Unused
Status[6]	Unused
Status[7]	Unused

Table 3.2: Status Register bits function

The *Status Register* is accessed by both the Python APIs and the bridge. In more detail, the Python APIs are used to set the required flags when data coming from the USB is complete and valid, such as a new instruction or a new address, while the bridge has the role of reset these flags when the data have been received by *X-HEEP*. The register reset operation was implemented without altering the interface of the register module. In fact, a simple bridge-driven multiplexer defines which data is sent to the register block. Under normal conditions, the data and address provided by the front-end block are passed directly to the register module. When the reset signal is active, however, the address bus is driven with the address of the corresponding register and the data bus with zeros, as shown in Listing 3.8.

```
always @(*) begin
    if (~rst_new_addr_valid_to_regs) begin
      reg_address = `REG_BRIDGE_STATUS;
      write_data  = bridge_status_usb & ~(8'b00000100);
      reg_write   = 1'b1;
    end

    else if (~rst_instr_valid_to_regs) begin
      reg_address = `REG_BRIDGE_STATUS;
      write_data  = bridge_status_usb & ~(8'b00000010);
      reg_write   = 1'b1;
    end

    else begin
      reg_address = reg_address_fe;
      write_data  = write_data_fe;
      reg_write   = reg_write_fe;
    end
end
```

Figure 3.8: SystemVerilog description of the status register reset mechanism.

As described in previous sections, the registers module and the bridge operate in different clock domains. For this reason, synchronisers and a handshake protocol have been implemented to ensure flawless asynchronous communication between these two regions.

The CW305 register definitions were also modified according to the project requirements, as was the register parallelism.

The entire system, including the bridge, the microcontroller and the CW305 wrap, was simulated by means of *Verilator* using a testbench very similar to the one already used for the software model simulation. This time, however, two different clock signals had to be generated for both the USB and the *X-HEEP* clock domains, in order to ensure that no communication errors occurred due to synchronism problems.

In addition, it was also necessary to simulate the pseudo-USB communication to the top-level CW305 to test the effectiveness of the status register in synchronising the transmission of new data from the host PC when the bridge was already busy in a transaction with X-HEEP.

The entire test bench can be summarised as follows:

– first the system is reset and initialized.

– Next, a new instruction or address is read from the firmware file. The testbench then checks the value of the status register. If the instruction valid flag is zero, it sends the new data to the board. Once the data has been sent, the instruction valid flag is set by writing the entire status register, after which USB communication stops.

– The bridge then polls the status register until the instruction valid flag equals 1. When this happens, communication with *X-HEEP* begins. Once *X-HEEP* has received the data, the bridge signals the reset of the status register flags and waits for them to return to zero.

– The host PC is then free to send new data.

When the firmware loading procedure is complete, the host PC signals to write a `1` to the address `2000000C`, which triggers the *X-HEEP* exit boot loop condition.

Simulation extracts are shown in the Figure 3.9 and Figure 3.10. In Figure 3.9 it is possible to notice the pseudo-USB communication between the PC and the board. In particular, the synchronisation mechanism between the board and the host PC can be observed. The latter in fact continuously checks the value of the status register. When the address valid flag is 0, the PC proceeds to send a new instruction and set the instruction valid flag, then waits for this flag to be reset.

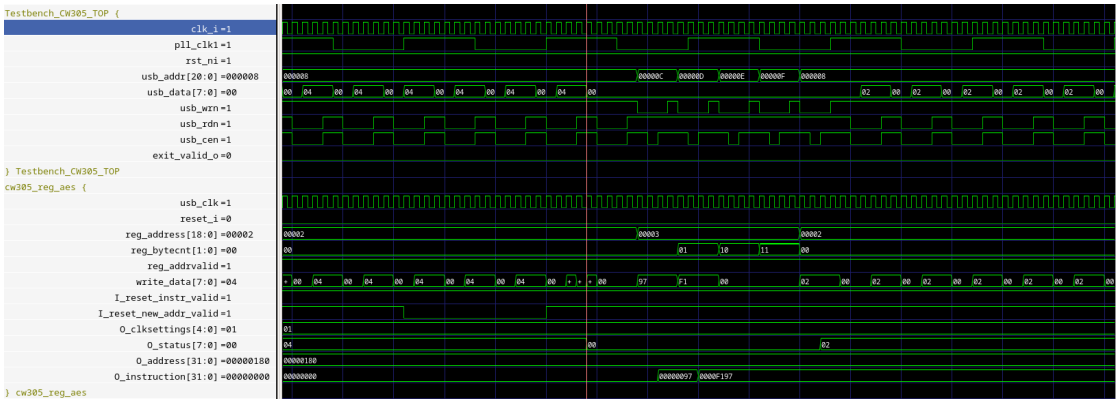


Figure 3.9: CW305 Simulation - USB communication

Figure 3.10, on the other hand, shows the OBI transaction between the bridge and *X-HEEP*. Once the instruction valid flag is set, the request to the microcontroller is sent from the bridge. The bridge then proceeds to reset the instruction valid flag when the grant signal is asserted by *X-HEEP*.

Figure 3.11 shows the exit boot loop procedure. An OBI request to the address `2000000C` is sent by the bridge. After some clock cycles, the `EXIT_LOOP` flag is raised.

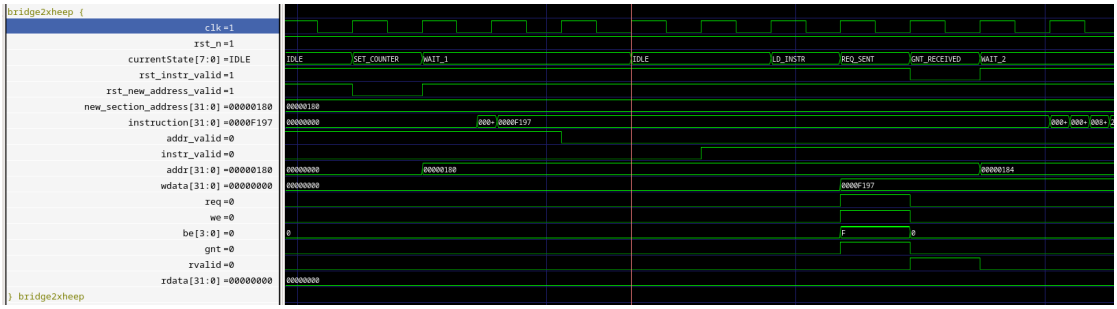


Figure 3.10: CW305 Simulation - OBI transaction

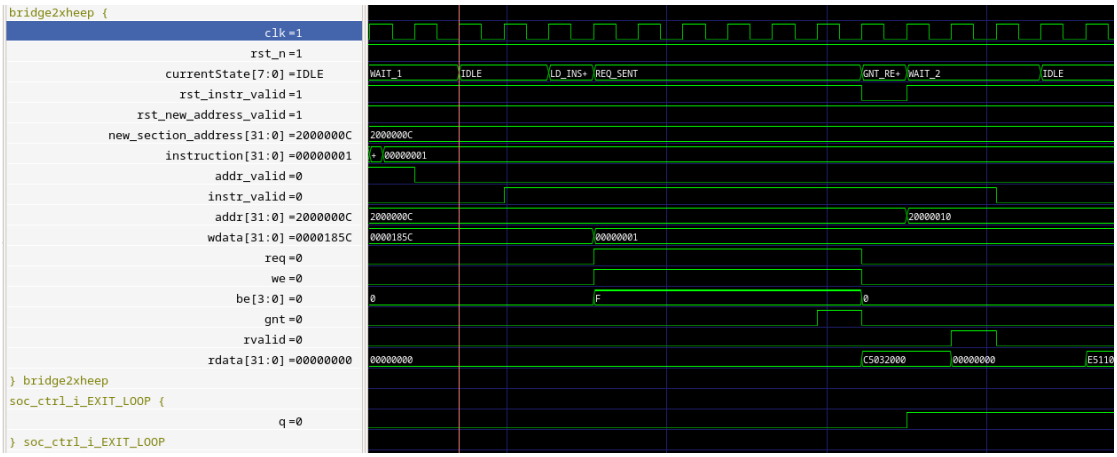


Figure 3.11: CW305 Simulation - Exit Boot Loop Flag

A simple *blink* program was run on the microcontroller to verify that the firmware loading procedure was completed correctly. After the exit boot loop flag is set, the microcontroller begins the program execution and the LED3 starts blinking, as it can be observed in Figure 3.12:

3.3 Synthesis on FPGA

Since the on-board FPGA is a Xilinx Artix-7, the entire project was synthesised using the Vivado Design Suite tool. A specific core file was designed to enable automated synthesis using FuseSoC [24].

Some changes had to be made to the default I/O pad configuration for *X-HEEP*. The design features a pad ring through which the microcontroller's I/O pins can be accessed; however, Vivado only allows synthesis of such a structure for signals connected to the FPGA's physical pins. Therefore, to avoid synthesis errors, the unused I/O signals were removed from the pad ring and the rest were connected to the physical pins on the board. In particular, the microcontroller's `GPIO 2` was

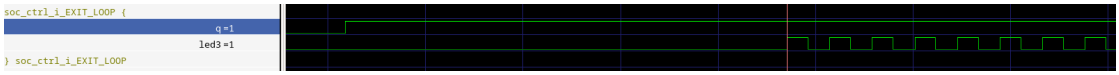


Figure 3.12: CW305 Simulation - LED blinking

connected to the on-board `LED3`, thus enabling verification of the system's correct operation through the same blink program used for the simulations. `GPIO 3` was also connected directly to *Status Register* bit 3, and `GPIO 4` was mapped to a specific FPGA pin for triggering the oscilloscope data acquisition.

3.4 On-board test

As in the simulation, the initial algorithm tested on the board was a simple LED blink program. In parallel, the functionality of the microcontroller's UART peripheral was also verified. Its TX and RX lines were mapped to specific FPGA pins, allowing communication with a PC via a UART-to-USB converter. Data transmitted from the microcontroller was received and displayed on the PC using the command-line tool `picocom`. The algorithm specifically toggled `GPIO 2` pin—connected to `LED 3` on the board—and sent the updated pin status over UART.

Once the correct functioning of the device was verified, it was ready to perform the side channel analysis of encryption algorithms.

Chapter 4

Power Analysis Side Channel Attack

4.1 Introduction

As mentioned in the introduction to this thesis, all electronic devices have power consumption during their operation due to various contributions, as shown in the Equation 4.1:

$$P_{tot} = P_{sc} + P_{static} + P_{dynamic} \tag{4.1}$$

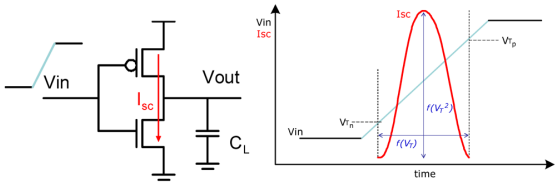


Figure 4.1: Visualization of the short-circuit current for a logic inverter.

The first contribution relates to the *short-circuit current*. Considering the structure of the logic inverter built with CMOS technology, as shown in the Figure 4.1, it can be seen that, once a logic state has been reached, one of the two transistors is in a non-conductive mode. Consequently, current cannot flow from V_{DD} to ground. However, for a brief moment during commutation when V_{IN} reaches $V_{DD}/2$, both transistors are in conduction, creating a direct path between the supply and ground. This causes the short-circuit current.

The second contribution is due to the parasitic *leakage currents* that exist due to the non-ideality of transistors. Ideally, transistors should behave like perfect

switches that prevent any current from flowing when they are turned off. In reality, transistors are slightly conductive even when turned off due to parasitic effects such as short-channel effects, drain-induced barrier lowering (DIBL) and sub-threshold current. While these effects were once negligible, they have become increasingly prominent as technology has advanced and transistor dimensions have decreased.

Nevertheless, the primary source of information regarding power side-channel attacks is the last term.: the **dynamic power dissipation**.

Dynamic power dissipation is caused by the charge and discharge of the load capacitance during the commutations. For the sake of clarity, during the $0 \rightarrow 1$ transition the hi-side pMOS is conducting, creating a connection between the power rail and the load capacitance, which allow the capacitor to charge up to V_{DD}. This implies an energy transfer from the power rail to the capacitor and so power consumption. The energy stored in the capacitor is then transferred to ground during the $1 \rightarrow 0$ transition.

Dynamic power dissipation can be modelled as described in Equation 4.2:

$$P_{dynamic} = \alpha \cdot V_{DD}^2 \cdot C_L \cdot f_{clk} \tag{4.2}$$

where V_{DD}, C_L, f_{clk} and α represent the supply voltage, the load capacitance, the clock frequency and the switching activity factor, respectively.

From the Equation 4.2 it is clear that the dynamic power consumption in proportional to the switching activity factor α, which in turn depends on what algorithm the device is executing. In other terms, it is possible to find a correlation between the internal state of the device and the its power consumption. This is the key idea behind *Power Side-Channel Attacks*.

The measurement of power traces is also quite simple and inexpensive. The typical measurement setup in fact only requires inserting a shunt resistor of reasonably small value (typically in the order of a few tens of ohms) in hi-side configuration between the power line and the supply pin of the device, as shown in Figure 4.2. In this way, by measuring the voltage drop at the ends of the resistor, it is possible to determine the instantaneous current absorption, from which it is possible to derive the power consumption according to the expression $P_{device} = V_{DD} \cdot I_{device}$. Note that the supply voltage is assumed to be constant during the analysis. A similar measurement setup is also used for electromagnetic (EM) side-channel attacks. In that case, however, it is not necessary to insert a shunt resistor on the power supply network as the current consumption is measured directly through a current probe, represented in its simplest form by a coil of wire.

Since the value of V_{DD} is fixed, any variation in the current absorption is translated to a variation in the power consumption. In particular, for the hi-side configuration, an higher current absorption generates a bigger voltage drop across

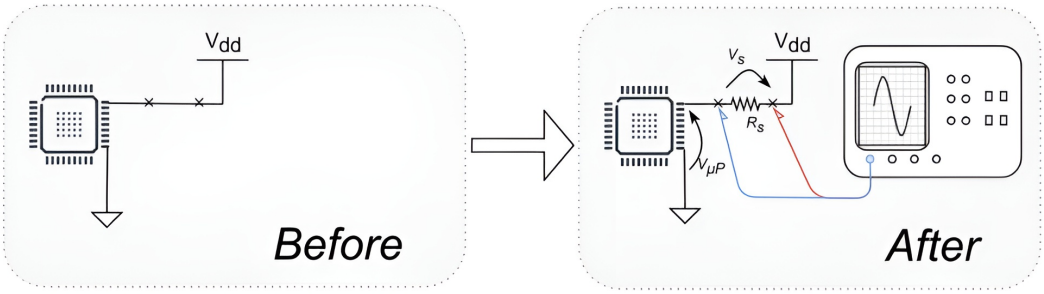


Figure 4.2: Implementation of the shunt resistor as current sensor.

the shunt resistor. As result, the trace measured by the oscilloscope will have a negative peak.

To highlight absorption peaks more strongly, the oscilloscope is usually coupled in AC mode to eliminate the DC voltage contribution from V_{DD}. Better measurement results are obtained if the shunt resistor is connected to the pin which powers the cryptographic core, especially if the device has multiple supply pins. Removing the decoupling capacitor from the power distribution network and bypassing any internal voltage regulator is also beneficial, since these tend to reduce the voltage variations that are useful for the attack.

By processing the power traces, it is possible to extract useful information about the internal state of the device, including the secret key employed by the encryption algorithms.

There are various types of power analysis attacks, each with its own effectiveness and complexity in implementation. The main ones are listed below:

– *Simple Power Analysis*

– *Differential Power Analysis*

– *Correlation Power Analysis*

4.2 Simple Power Analysis (SPA)

Simple Power Analysis (SPA) is a technique that involves the direct interpretation of power consumption measurements collected during cryptographic operations [8]. If the algorithm being executed is know by the attacker, it is quite simple to retrieve information about the internal processing state.

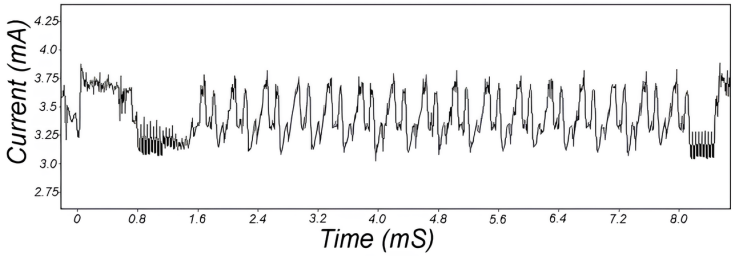


Figure 4.3: SPA trace showing an entire DES operation [8].

By means of SPA is therefore possible to determine the sequence of instructions executed, so it can be used to break cryptographic implementations in which the execution path depends on the data being processed. For instance, an algorithm with conditional jumps will have different power traces depending on whether the jump is performed or not. Possible examples are DES key schedule, DES permutations or comparisons, where typically a conditional branch is taken if a mismatch is found.

This principle can be illustrated more clearly with a simple password check program (C code below): if the password check is performed by comparing each character of the inserted string with the correct password, the algorithm's execution time (and thus the power traces) will differ depending on the number of correct characters, as shown in Figure 4.4 [25].

It can be seen that the calculation takes longer when a larger number of characters pass the comparison. Thus, it is possible to guess the correct password character by character, simply by looking at the resulting power traces, instead of guessing the entire password at once. For example, assuming the password contains only alphabetical letters, for a 4-character password it is only necessary to guess $26 + 26 + 26 + 26 = 104$ combinations instead of $26^4 = 456976$ ones.

```c
#include <stdio.h>
#include <stdlib.h>

int main(){
    char inserted_password[4];
    char correct_password[4] = {'a', 'b', 'c', 'd'};
```

```
7       int pass_ok = 1;
8
9       // Get user password
10      printf("Enter password: ");
11      scanf("%s", inserted_password);
12
13      // Check the password char by char
14      for (int i = 0; i < sizeof(correct_password); i++){
15          if (correct_password[i] != inserted_password[i]){
16              pass_ok = 0;
17              break;
18          }
19      }
20
21      if (pass_ok) {
22          printf("Password Correct.\n");
23      } else {
24          printf("Password Fail.\n");
25      }
26      return pass_ok;
27  }
```

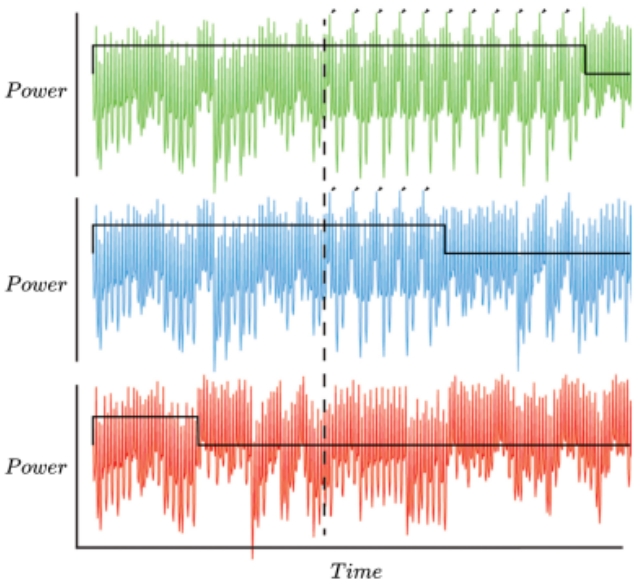


Figure 4.4: Possible power traces for totally correct, partially correct and incorrect passwords.

4.3 Differential Power Analysis (DPA)

The *Differential Power Analysis* (DPA) was the first statistics-based method applied to power-side channel analysis. It was introduced by Paul Kocher et al.[8] in their 1999 paper "Differential Power Analysis". Unlike SPA, the Differential Power Analysis method requires the capture of a large number of power traces to achieve key recovery by exploiting statistical methods. In fact, by averaging a large number of traces, even the smallest variation in the data can be revealed, as the noise cancels out.

The key to the success of this type of attack lies in the existence of a relationship between an intermediate state of the algorithm and a measurable physical quantity (e.g., power consumption). This intermediate state is typically the result of an operation performed between the secret key and the known input data (the plaintext or nonce). From the known input data and the collected traces, it is possible to derive information about the secret key simply by analyzing which hypothesis of the key produces the measured power consumption for a given input data.

An important role is played by the *selection function*, which is essential for the computation of the intermediate state value used for the attack. An example of a selection function for cryptographic algorithms is the result of a XOR operation between the secret key and known data.

Traces are grouped according to the value of the bit computed by the selection function for a given key guess. Typically the target bit is the LSB, but the rule can be applied to any bit.

Once the traces have been partitioned into two groups, one in which the LSB of the intermediate state is equal to 0 and one in which it takes the value 1, the mean difference between the two groups can be calculated. If the hypothesis is correct, the difference between the two groups will exhibit spikes, indicating that there is a difference in power consumption due to the presence of a different number of bits equal to 1. Otherwise, the two groups contain only noise and their difference will tend to zero out as the number of analysed traces increases.

The Python code for the DPA algorithm is shown in the Listing 4.5.

One detail to pay attention to when working with DPA is the possible conditioning of traces before the actual analysis. Indeed, it may be necessary to remove some samples to improve the signal-to-noise ratio or realign the traces. Due to uncertainties about the device clock (jitter) or lack of synchronization between it and the oscilloscope used for measurements it could happen that the operation under attack is performed at different time instants. As a result, it might be more complicated to correlate the measured data with the algorithm model, thus reducing the effectiveness of the attack. An example of this situation is shown in Figure 4.6.

Figure 4.5: Python code example for the DPA algorithm.

```
TARGET_BIT = 0
difference_of_means = []

for key_guess in range(8):
    traces_0 = []
    traces_1 = []
    for i in range(len(nonces)):
        hypothetical_output = selection_function(key_guess, known_input,
        ↪  TARGET_BIT)

        if hypothetical_output == 0:
            traces_0.append(traces[i])
        else:
            traces_1.append(traces[i])

    diff = np.mean(traces_1, axis=0) - np.mean(traces_0, axis=0)
    difference_of_means.append(diff)
```

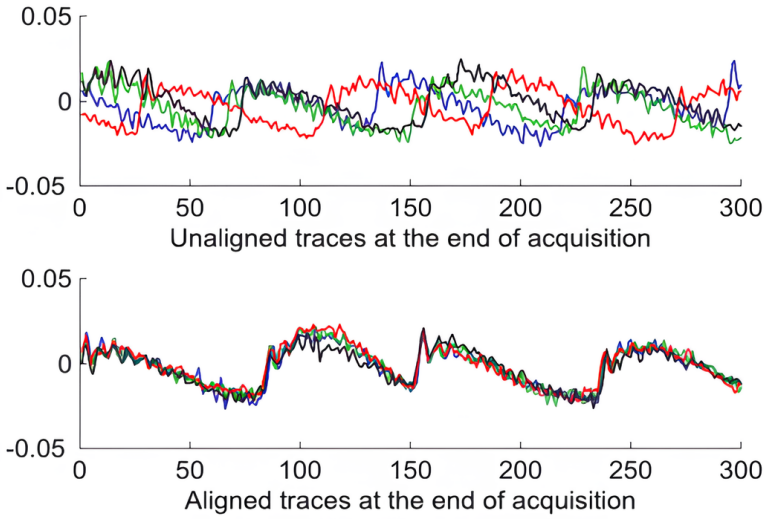


Figure 4.6: Difference between aligned and unaligned power traces

4.4 Correlation Power Analysis (CPA)

Similar to DPA, *Correlation Power Analysis* (CPA) is a type of attack that is not limited to the pure observation of acquired power traces, but involves using statistical methods to retrieve the content of the entire secret key used by the encryption algorithm. It was introduced by Eric Brier, Christophe Clavier, and Francis Olivier in their CHES 2004 paper "Correlation Power Analysis with a Leakage Model" [26] and it can be considered an extension of the DPA.

In the previous sections, it has been explained that the main cause of power consumption of electronic devices is related to the charging and discharging of load capacity C_L. It is therefore evident that the greater the number of '1's present on the bus, the greater the energy absorbed by the power supply network to charge the C_L capacity. The number of '1's present on the bus, also referred to as the *Hamming Weight*, has an influence on power consumption. In other words, it is possible to prove the existence of a correlation between the *Hamming Weight* and consumption peaks, and this is the basic idea behind this attack. More in general, if a word D contains m independent and uniformly distributed bits, the average hamming weight and variance would be $\mu_{HW} = \frac{m}{2}$ and $\sigma^2_{HW} = \frac{m}{4}$

The *Hamming Weight* is not the only possible leakage model that can be used for this type of attack. In the paper by Eric Brier, Christophe Clavier and Francis Olivier [26] they proposed an approach based on the *Hamming Distance* model, which relates the power consumption to the number of bits toggling on the bus when transitioning from a value to the successive one. Assuming that the initial state of a bus is a constant m-bit word R, unknown but not necessarily zero, the Hamming Distance between D and R can be described as $H_D(D \oplus R)$, where the $\oplus$ symbol denotes the bitwise XOR operation. If the word D is a random value with a uniform distribution, the same is true for $D \oplus R$, so $HD(D \oplus R)$ has the same average value $\mu_{HW} = \frac{m}{2}$ and variance $\sigma^2_{HW} = \frac{m}{4}$ as for the case of the Hamming Weight.

For instance, the transition between `0x51` to `0x53` denotes a Hamming Distance $H_D(\texttt{0x51} \rightarrow \texttt{0x53}) = H_D(\texttt{0x51} \oplus \texttt{0x53}) = 1$, since only 1 bit toggles during the transition.

Knowing the hamming distance, it is possible to find a linear relationship between it and the power consumption W, as shown in the Equation 4.3, where a represents the scalar gain between the hamming distance and the power consumed and b is a term that takes into account all parasitic effects that have an effect on measurements such as offset or noise.

$$W = a \cdot H_D(D \oplus R) + b \tag{4.3}$$

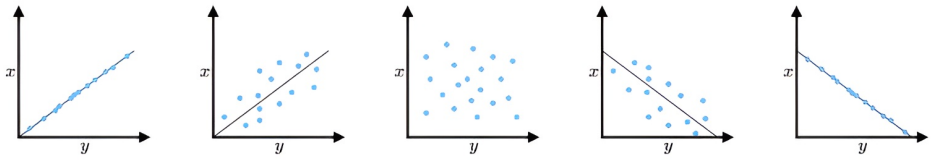
The linear relationship proposed by the authors [26] can be seen as a simplification since it does not take into account many contributions that influence power consumption, but it is by no means unrealistic as bus lines are typically the main cause of consumption in embedded systems. As a consequence of the linear relationship, the following expression for the variances is also verified (Equation 4.4):

$$\sigma_W^2 = a^2\sigma_{H_D}^2 + \sigma_b^2 \tag{4.4}$$

The *Correlation Power Analysis* is based on the idea that a linear dependency exists between the power consumption and the Hamming Weight or the Hamming Distance of an algorithm intermediate value. The *Pearson's Correlation Coefficient* ρ is used to prove this linear dependency, as shown in Equation 4.5.

This coefficient always assumes values in the range [-1, 1] and, depending on its value, it has a different meaning: a correlation equal to 1 means that a direct linear relation exist between the two variables (an increase in the former corresponds to an increase in the latter), while a correlation equal to -1 indicates an inverse linear relation (an increase in the former corresponds to an decrease in the latter). More in general, positive values of correlation denote a direct linear trend, vice versa for the negative ones. If no relationship exist, the correlation assumes the value 0. A graphical summary is shown in Figure 4.7.

$$\rho_{WH_D} = \frac{cov(W, H_D)}{\sigma_W \sigma_{H_D}} = \frac{a\sigma_{H_D}}{\sigma_W} = \frac{a\sigma_{H_D}}{\sqrt{a^2\sigma_{H_D^2} + \sigma_b^2}} = \frac{a\sqrt{m}}{\sqrt{ma^2 + 4\sigma_b^2}} \tag{4.5}$$



((a)) Perfect positive correlation. $\rho_{WH} = 1$

((b)) Low positive correlation. $0.3 > \rho_{WH} > 0$

((c)) No correlation. $\rho_{WH} = 0$

((d)) Low negative correlation. $0 > \rho_{WH} > -0.3$

((e)) Perfect negative correlation. $\rho_{WH} = -1$

Figure 4.7: Visual representation of the correlation factor for different relationship between the x and y variables.

Having shown that a correlation exists between power consumption and internal state R, it is possible to use this tool to retrieve the secret key used by the algorithm. Given a set of input data D and relative power traces, correlation can be used to find the secret key R among all possible 2^m combinations by observing which one has the highest correlation with the measured traces.

Obviously, the effectiveness of this attack is greater when the parallelism of the data to be recovered is relatively small. This is the case, for instance, with the AES

algorithm. Although it is based on a 128-bit key, which requires testing all 2^{128} possible combinations in the classical way, each round is performed on 1 byte of data, so using CPA it is sufficient to test only the possible $2^8 = 256$ combinations to recover a key byte, a very simple task that can be performed by any modern computer in few minutes.

The general representation of a Correlation Power Analysis is shown in Figure 4.8. The standard attack procedure consist of few steps [27]:

1. Choose an intermediate variable of the algorithm as the attack point. This variable, called *selection function*, must be a function $f(d, k)$, of a part of the key k and the known non-constant data d (e.g. *plaintext*, *nonce*).

2. Measure the power consumption of the device while it executes the cryptographic algorithm l times. For each execution, the power trace is collected. At the end of the process, all power traces will form a matrix $\mathbf{T}$ of size $l \times s$, where s is the number of samples. It is important to verify that power traces are correctly aligned in time and to note of the input data $\mathbf{d} = (d_1, ..., d_l)$ used as *known-data*.

3. Calculate hypothetical intermediate values for every possible candidate of k. Let $\mathbf{k} = (k_1, ..., k_p)$ be the vector of p possible candidates for k, also usually referred to as key hypotheses or key guesses. For each key guess, the selection function $f(d, k)$ is exploited to compute the hypothetical intermediate values corresponding to the vector $\mathbf{d}$. Performing this calculation for all key hypotheses results in a matrix of hypothetical intermediate values, denoted by $\mathbf{V}$, of size $l \times p$.

4. Map hypothetical intermediate values to hypothetical power consumption values. A leakage model must be chosen to estimate the power consumption (*i.e.*, hypothetical power consumption) exposed by the device when processing a value. The *Hamming Weight* and the *Hamming Distance* are often used as leakage models. Each value in $\mathbf{V}$ is then mapped to a corresponding hypothetical power consumption value, resulting in a hypothetical power consumption matrix $\mathbf{H}$ of size $l \times p$.

5. Compare the hypothetical power consumption values with the power traces. As explained above, the the Pearson's correlation coefficient is used to examine the linear correlation between the hypothetical power consumption values of each key candidate with the measured traces at every position. Specifically, the correlation coefficient is computed between each column h_i of the matrix $\mathbf{H}$ and each column t_j of the matrix $\mathbf{T}$, resulting the element $r_{i,j}$ of the matrix $\mathbf{R}$ of size $p \times s$, where $r_{i,j}$ is computed as:

$$r_{i,j} = \frac{\sum_{u=1}^{l}(h_{u,i} - \overline{h_i})(t_{u,j} - \overline{t_j})}{\sqrt{\sum_{u=1}^{l}(h_{u,i} - \overline{h_i})^2}\sqrt{\sum_{u=1}^{l}(t_{u,j} - \overline{t_j})^2}} \tag{4.6}$$

6. The key can be recovered based on the fact that the higher value $r_{i,j}$ indicates a stronger linear correlation between the columns h_i and t_j, suggesting a better match under the assumed leakage model. Let c_k be the index of the correct key k_{ck} (i.e., the key that is used in the device) in the vector $\mathbf{k}$, and c_t be the index of the power consumption values t_{ct} that depend on the intermediate values v_{ck}. The columns h_{ck} and t_{ct} should be strongly correlated. Thus, the highest value $r_{ck,ct}$ in the matrix $\mathbf{R}$ reveals the indexes of the correct key c_k and the position c_t.

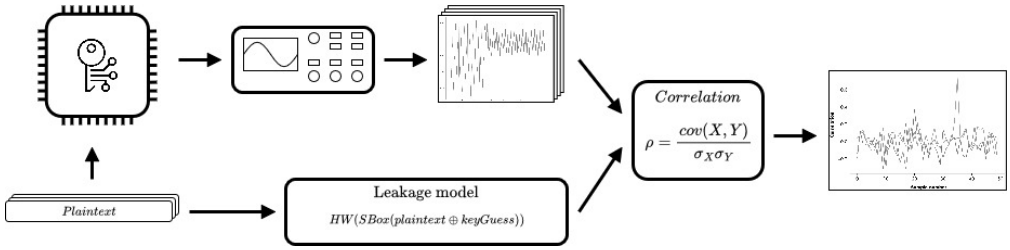


Figure 4.8: Schematic of the measurement setup for the CPA attack

4.5 Countermeasures

Because the effectiveness of side-channel attacks lies in the leakage of computation-related information through unintended channels, several countermeasures have been introduced to eliminate or otherwise reduce this leakage of information. These countermeasures fall in three main categories:

- **Refactoring the algorithm**: these countermeasures try to rewrite the algorithm in a way that no information is leaked on the side channel
- **Eliminate the information**: the idea behind these countermeasures is that of trying to reduce the correlation between the data being processed and the measurements taken on the side channel
- **Suppress the side channel**: countermeasures whose purpose is to physically suppress the side channel

4.5.1 Refactoring the algorithm

Countermeasures that fall into this category are intended to mitigate the possibility of SPA attacks. Indeed, simple power analysis exploits the dependency between the control flow and the value of the secret key, as shown in the example in Section 4.2, where the password check is performed by comparing one character at the time.

To eliminate this dependency, it is sufficient to rewrite the algorithm by choosing an implementation that makes use of constant-time execution paths. This can be achieved, for example, by eliminating all conditional constructs (e.g., if statements) by turning them into predicated instructions. Some ISAs support predicate instructions natively.

4.5.2 Eliminate the information

Preventing side channel attacks based on statistical methods, such as DPA and CPA, is more challenging than the SPA case and requires some caveats. In fact, the effectiveness of these attacks is based on their ability to isolate the useful signal from the noise by simply analyzing a good number of traces and looking for temporal correlation between the measured data and the algorithm model. Consequently, one possible countermeasure against this type of attack may be to **hide the sensitive operation in time** by introducing a non-deterministic delay at the instant the operation is actually performed. In this way, the collected traces will not be aligned with respect to the sensitive operation, lowering the correlation of instantaneous power consumption. Hiding in software can be easily achieved introducing a loop of NOP operations in the algorithm, where the number of

iterations is picked at random each time. The same can be done in hardware by introducing small disturbances on the clock generator or by using different clock generators.

Despite being a simple solution, this countermeasure does not *entirely* eliminate the correlation on the data: by collecting more traces it will still be possible to find a correlation between the model and power consumption.

Masking is another widely adopted countermeasure against *Differential Power Analysis* (DPA) and, more generally, side-channel attacks. Its core idea is to randomly divide each sensitive intermediate variable involved in a computation into $(d + 1)$ *shares*, where d, the *masking order*, acts as a security parameter [28]. The purpose of this approach is to ensure that observing fewer than all the shares reveals no information about the original secret value.

By concealing the actual input values used during operations, masking prevents attackers from accurately predicting intermediate values in the cryptographic algorithm. In practice, masking can be implemented by adding a random value to each share using an XOR operation. This randomization does not alter the true input data or secret key, since most cryptographic algorithms already use XOR-based key additions that naturally cancel out the added randomness.

Despite its effectiveness, masking introduces significant computational and hardware overhead, which limits its applicability in resource-constrained environments. In particular, hardware implementations often require additional components such as *GF multipliers*, used, for example, in *Domain-Oriented Masking* (DOM), whose number increases quadratically with the masking order, as expressed in Equation 4.7:

$$GF\ Multipliers \propto (d+1)^2 \tag{4.7}$$

4.5.3 Suppress the side channel

These countermeasures aim to eliminate the side channel completely, thus removing the source of information. An example can be the use of differential logic styles (such as WDDL [29]). With this kind of logic, each gate computes both the actual and complementary output of the implemented function (e.g. both x and $\overline{x}$). As result, the switching activity of the gate is kept constant during the computation, thus removing the dependency between the power consumption and the number of gates switches.

Chapter 5

Advanced Encryption Standard (AES)

Among the encryption algorithms tested on the microcontroller is the *Advanced Encryption Standard* (**AES**), the encryption specification for electronic data adopted by the U.S. National Institute of Standards and Technology (NIST) in 2001. [30] [31]

AES was derived as a variant of the Rijndael block cipher developed by the two Belgian cryptographers, Joan Daemen and Vincent Rijmen, from which it differs in block size and key length. Indeed, AES specifications allow a block size of 128 bits only, but three different key lengths: 128, 192 and 256 bits.

The algorithm described by AES is a symmetric-key encryption one, where the same key is used for both the encryption and decryption procedure, and it consist of *rounds*, sequences of simple operations iterated several times until the whole *plaintext* is encrypted. In particular, the number of rounds performed by the algorithm depends on the *key* length: 10 rounds for 128-bit keys, 12 rounds for 192-bit keys, 14 rounds for 256-bit keys.

Of course, each of these operations can be undone knowing the secret key, so that authorised personnel can retrieve the encrypted information.

AES is based on a design principle known as the substitution-permutation network, in which the *chipertext* is obtained as a result of successive byte substitution and permutation operations performed on the *plaintext*, which is organized as a 4x4 byte array. In addition to these, operations involving the use of the secret key are executed in order to make it impossible to decipher the result by reversing the sequence of operations performed unless the key is known.

The sequence of operations performed in each *round* is shown in the Figure 5.1 and listed hereafter.

- *KeyExpansion*: round keys are derived from the secret key using the AES key schedule;
- *AddRoundKey*: a bitwise XOR operation is performed between each byte of the *plaintext* (or *State*) and a byte of the round key;
- *SubBytes*: each byte of the *State* array is replaced with another one obtained in a non linear way by means of a 8-bit substitution box (*S-Box*);
- *ShiftRows*: bytes in each row of the *State* array are shifted in a circular way by a certain offset. This step is necessary to avoid the columns being encrypted independently;
- *MixColumns*: the 4 bytes of each column are combined together by means of a linear combination, producing 4 different bytes in a reversible way.

It is important to notice that all the rounds consist of the same sequence of operations except for the last one, where the *MixColumns* phase is absent.

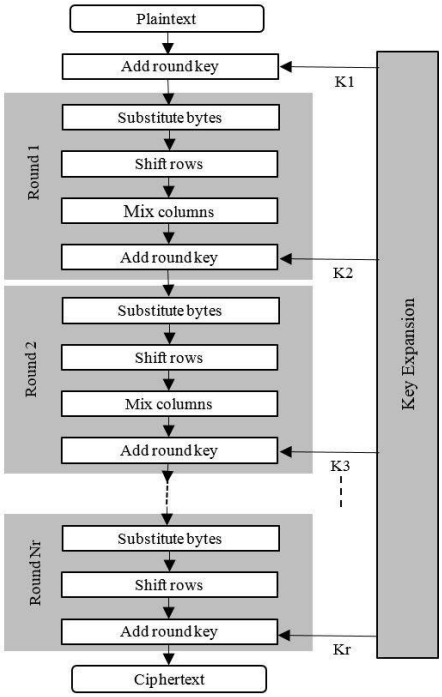


Figure 5.1: Visualization of the *AES* algorithm. [32]

The simplicity of the AES algorithm makes it very fast in both hardware and software implementation, which is why, combined with its inherent mathematical robustness, it is used as the standard encryption algorithm.

In fact, even using a 128-bit key it would be necessary to test $2^{128} \simeq 3.40 \cdot 10^{38}$ possible combinations to discover the secret key. Even assuming a modern computer can test 1 billion combinations per second, it would still take $3.4 \cdot 10^{29}$ seconds to guess the correct key, or approximately 10^{22} years. As a metric for comparison, the estimated age of our universe is around 13 billion years (10^{10}), so it is clear that the algorithm is mathematically impossible to bypass.

5.1 AES Robustness against Side-Channel Attacks

Despite its mathematical robustness, the AES algorithm is susceptible to side-channel attacks and, more in particular, to *Power Side-Channel Attacks*, as explained in Section 4.4.

More specifically, the greatest vulnerability of AES lies in the structure of the S-Box. In fact, AES's S-Boxes are designed to introduce non-linearity and confusion into the encryption process, in order to make the algorithm robust against traditional cryptanalysis methods. As a side effect, the non-linearity introduced by the S-Box causes a large number of bits to be toggled, which in turn leads to a large variation in power consumption. This variation can be measured to recover information regarding the secret key.

For this reason, the Hamming Weight or the Hamming Distance at the S-Box output are typically used as leakage models. In each round, a bitwise XOR operation is performed between a key byte and a plain text byte, whose result is then used to point to the S-Box, a look-up table that returns the data corresponding to the requested address. The output of the S-Box is therefore closely related to the secret key, and this is where the information leaks the most.

The purpose of this analysis is therefore to verify whether a modification to the structure of the S-Box, like a reduction of its non-linearity, can reduce the sensitivity of the algorithm to side channel attacks. This type of solution is, at least in theory, very promising since it allows for the implementation of a simple and effective countermeasure against side channel attacks at almost no cost.

5.2 Measurement Setup

The analysis was conducted on the *ChipWhisperer* CW305 board as shown by Figure 5.2, where the *X-HEEP* microcontroller is synthesized as the target platform, and all measurements were performed using a *Picoscope PS5000* oscilloscope. The scope configuration is the following:

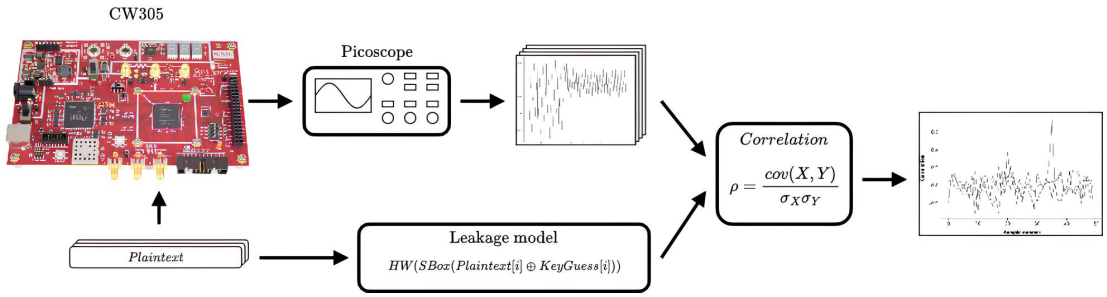


Figure 5.2: Measurement setup for the CPA attack on AES

- Channel A: used for trace acquisition. AC coupled to remove the unwanted 3.3 V DC component of the power supply.
- Channel B: used as the trigger source. DC coupled.
- Sampling Frequency: 125 MHz. This sampling frequency value was chosen to be much higher than the microcontroller clock frequency (10 MHz) in order to compensate for any synchronization problems caused by the different clock sources used by the oscilloscope and the microcontroller.
- Resolution: 8-bit

As explained in Section 1, side-channel analysis was performed on the software implementation of AES. More specifically, a slightly modified version of the C implementation[1] of AES was the subject of the analysis. The main changes consisted of the addition of the code necessary for the generation of the trigger for the oscilloscope. In fact, the trace acquisition is triggered only at the first round of AES, even though the algorithm is executed in its entirety. More in detail, the acquisition procedure is performed as follows:

1. When a new encryption operation is going to be started, the bit 3 of the status register is set using the Python APIs provided by *ChipWhisperer*. Bit 3 of the status register is directly connected to the microcontroller's GPIO 3.
2. The microcontroller polls the value of GPIO 3 and waits until the PIN value becomes high.
3. The microcontroller sets GPIO 4 to 1. This PIN is connected to one of the physical pins of the FPGA, which also has a connector on the board. The oscilloscope probe for channel B is connected to this PIN and is used as the trigger source.

[1] `https://github.com/kokke/tiny-AES-c`

4. The microcontroller executes the algorithm. When execution is complete, the microcontroller waits for the GPIO 3 value to return low before starting a new encryption operation.

5. Bit 3 of the status register is reset via the Python API. At this point, a new operation can start.

This sequence of operations can be performed automatically as many times as desired using a Python notebook. For this analysis, several sets of 5,000 traces were collected, in each of which the algorithm was executed with a different S-Box.

5.3 CPA Attack

A *Correlation Power Analysis* attack was carried out on each set of traces using the *ChipWhisperer* framework. The Hamming weight at the S-Box output was chosen as leakage model for the attack, as shown in Equation 5.1.

$$leakage[byte_i] = HW(SBox[plaintex[byte_i] \oplus key[byte_i]) \tag{5.1}$$

Once the leakage model has been defined, the *ChipWhisperer* framework performs the entire analysis by carrying out the CPA attack on the 16 bits of the key. In particular, the correlation is calculated using an incremental algorithm, which allows the traces to be analysed in groups without losing information on the analysis of previous traces. The size of the groups defines the resolution used for the attack.

This makes it possible to follow the progress of the attack and assess whether or not it is successful. In addition, the framework also sorts the key guesses according to their correlation value and calculates some parameters useful for side channel analysis, such as Partial Guessing Entropy (PGE) and correlation graphs as a function of the number of traces analysed.

Figure 5.3 shows the shape of the power traces captured during the first round of the algorithm. From this data, it is already possible to obtain some information about the algorithm being executed. In particular, it is possible to distinguish the SubBytes phase at the beginning, ShiftRows in the middle, and MixColumns at the end.

A closer look at the SubBytes phase shows exactly 16 peaks, each relating to an operation performed on a byte of the state, confirming the presence of a 128-bit block cipher. Furthermore, the presence of 3 peaks in the shift rows phase indicates that 3 of the 4 rows of the state matrix have been rotated as expected, as the first row is not modified in this phase.

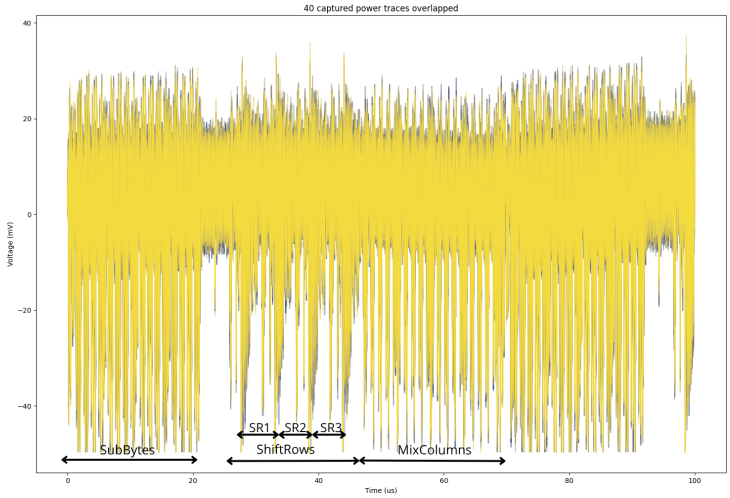


Figure 5.3: Power traces for the first round of AES.

A summary of the results obtained from the attack with each S-Box is available in Table 5.1 and Table 5.2. Chapter 7 will explore the meaning of the metrics shown in the table in more detail, considering both resistance to classical cryptanalysis and resistance to side-channel attacks.

SBox	Power analysis resistance		Cryptanalysis resistance		
	PGE < 10		**Non**	**Differential**	**Confusion**
	AES HW	AES SW	**Linearity**	**Uniformity**	**Coefficient**
$Rijandael$	$\sim$ 800 traces	$\sim$ 500 traces	112.0	4/256	0.111
$Freyre_1$	$\sim$ 1190 traces	$\sim$ 760 traces	100.0	8/256	4.500
$Freyre_2$	$\sim$ 2000 traces	$\sim$ 1300 traces	100.0	8/256	4.492
$Freyre_3$	$\sim$ 1340 traces	$\sim$ 400 traces	102.0	8/256	1.934
$Ozkaynak_1$	$\sim$ 1750 traces	$\sim$ 350 traces	106.7	10/256	0.103
$Hussain_6$	$\sim$ 1290 traces	$\sim$ 360 traces	112.0	4/256	0.111

Table 5.1: Trade-off between the cryptanalytic properties and PGE threshold in AES

The results of the attack showed that the type of S-Box used in the algorithm

Sbox type	NL min	NL max	DU	CCV	MCC	TO
Rijndael	112.0	112.0	4	0.1113	0.8125	7.8600
Freyre 1	100.0	110.0	8	4.5003	0.1328	7.5627
Freyre 2	100.0	110.0	8	4.4918	0.1289	7.5696
Freyre 3	102.0	110.0	8	1.9343	0.4492	7.6684
Hussain 6	112.0	112.0	4	0.1113	0.8125	7.8600
Ozkaynak 1	86.0	110.0	12	0.1030	0.7539	7.7983
Azam 1	94.0	110.0	10	0.0903	0.7656	7.8272
Azam 2	90.0	108.0	12	0.1105	0.7812	7.8243
Azam 3	94.0	110.0	10	0.1375	0.6875	7.8127

Table 5.2: Cryptographic properties of the S-boxes in exam for AES. NL: Non-Linearity. DU : Differential Uniformity. CCV : Confusion Coefficient Variance. MCC : Minimum Confusion Coefficient. TO : Transparency Order

can have an impact on sensitivity to side channel attacks. In fact, in the case of the standard S-Box (*rijandael*), approximately 500 traces are needed to fully recover the secret key, as shown in Figure 5.4. In the case of the *Freyre-2* S-Box, however, data shows that the same amount of traces traces is still not enough to recover all the bytes of the secret key (Figure 5.5).

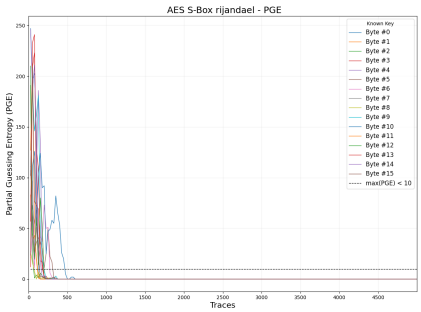


Figure 5.4: Partial Guessing Entropy for the standard AES S-Box.

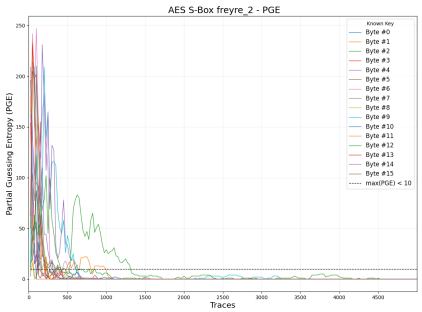


Figure 5.5: Partial Guessing Entropy for the S-Box Freyre 2.

As further confirmation, Figure 5.6 and Figure 5.7 show the success rates of attacks using the same number of traces for the two S-Boxes. Clearly, in the first case, a success rate of 1 is achieved after a few hundred traces, while for the Freyre-2 S-Box, some incorrect bytes are ranked first even with a larger number of traces,

reducing the success rate value.

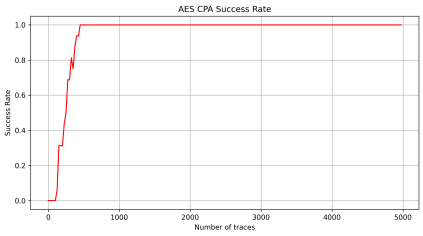


Figure 5.6: Success rate for the standard AES S-Box.

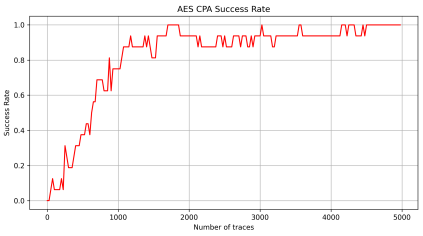


Figure 5.7: Success rate for the S-Box $Freyre_2$.

As for the remaining S-Boxes, only Freyre 1 has some minor advantages over the standard Rijndael S-Box, while Freyre 3, Hussain, and Ozkaynak perform as well as, if not worse than, the default AES S-Box. The PGE plots for these S-Boxes are shown in Figure 5.8, Figure 5.9, Figure 5.10 and Figure 5.11. Note that for the last three S-Boxes, the horizontal scale has been reduced to allow for better visualisation.

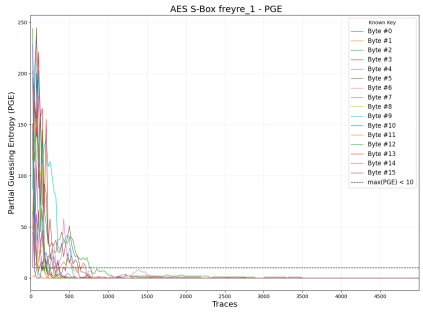


Figure 5.8: Partial Guessing Entropy for the S-Box Freyre 1.

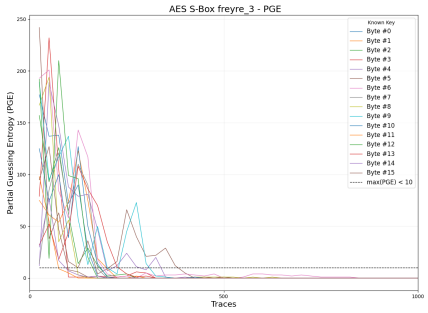


Figure 5.9: Partial Guessing Entropy for the S-Box Freyre 3.

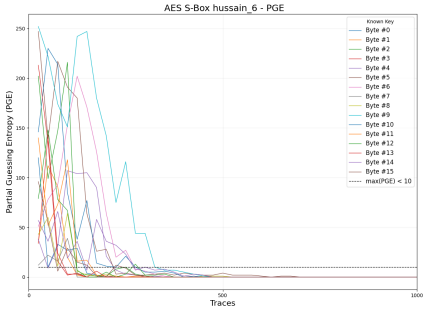


Figure 5.10: Partial Guessing Entropy for the S-Box Hussain.

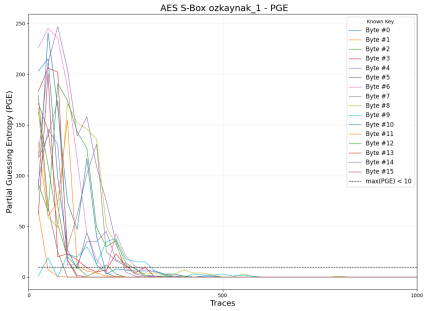


Figure 5.11: Partial Guessing Entropy for the S-Box Ozkaynak.

5.4 Considerations

The results show that modifying the structure of the S-Box can affect the algorithm's sensitivity to power side-channel attacks.

In this case, however, the advantages of this countermeasure are minimal, most likely due to the nature of the algorithm itself. For AES, the 128-bit key is used in groups of one byte each, making it very easy to implement an efficient CPA attack, as one byte of information is leaked each time. In fact, the signal-to-noise ratio is much higher than that found in the case of ASCON, which is described in the next chapter, where the leaked information relates to just one bit.

The SNR plots for the *Rijandael* and *Freyre 2* are shown in Figure 5.12 and Figure 5.13.

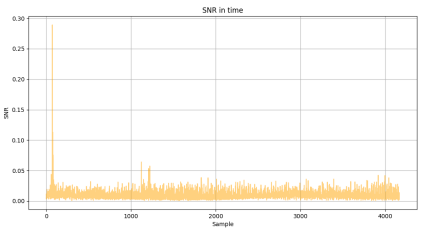


Figure 5.12: SNR for the Rijandael S-Box.

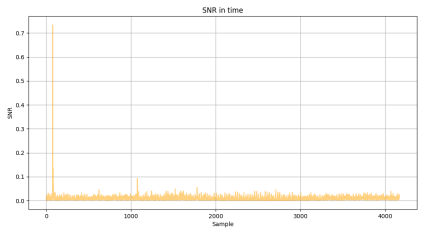


Figure 5.13: SNR for the Freyre 2 S-Box.

Despite the advantages in terms of side-channel resistance, this type of modification usually affects the algorithm's mathematical robustness against classic cryptographic attacks. Therefore, there is a trade-off between mathematical robustness and sensitivity to power side-channel attacks.

Chapter 6

ASCON

ASCON is a family of lightweight algorithms designed to provide Authenticated Encryption with Associated Data (AEAD), a hash function, and two eXtendable Output Functions (XOFs) [33]. The Ascon family has been designed to be efficient in constrained environments, such as RFID tags, sensors, and smart cards, where hardware resources are often limited to execute the time-proven AES algorithm. For this reason, in 2023 the Ascon family had been selected by US National Institute of Standards and Technology (NIST) for future standardization of the lightweight cryptography [34].

The ability to encrypt data combined with the possibility of authenticating the message through a hash function makes the algorithm even more secure for the type of application for which it was designed. In fact, encrypting sensitive data is as important as ensuring that the message is not tampered with during transmission. In these cases, adding a TAG to the message is useful to guarantee its authenticity. The TAG can be generated through hash functions, i.e., functions that generate a fixed-size result from a much larger input. The peculiarity of hash functions is that they always generate the same result from the same input data. Consequently, to verify the authenticity of the received message, it is sufficient to check that the hash generated by the received message is identical to that of the sent message.

The main advantages of ASCON over other cryptographic algorithms are [33]:

- **Multiple functionalities**: the ASCON family was developed with the goal of reducing resource usage. For this reason, the same permutations are used to build multiple functionalities, allowing for the implementation of AEAD, hash, and XOF functionalities to share logic. If these functions had been developed independently, they would inevitably have required more resources.

- **Online and single-pass**: an important feature of ASCON is its ability to require only the i-th plaintext block to generate the i-th ciphertext block, in

addition to the key, nonce, and associated data. This way, only one pass over the data is required.

- **Inverse free**: all the ASCON functions are designed to use the underlying permutations in the forward direction. This simplifies the decryption process and reduces implementation costs, since there is no need to develop the inverse function for permutations.

6.1 Internal structure

This section provides an in-depth look at the ASCON encryption process. The algorithm takes as input the 128-bits key K, the 128-bits nonce N, the associated data $A_1, ..., A_s$ and the plaintexts $P_1, ..., P_t$, both in 64-bit blocks, and the 64-bit initialization vector IV. The 128-bit tag T and the chipertexts $C_1, ...C_t$, each of 64-bits, are generated as output. A graphical representation of the process is shown in the Figure 6.1.

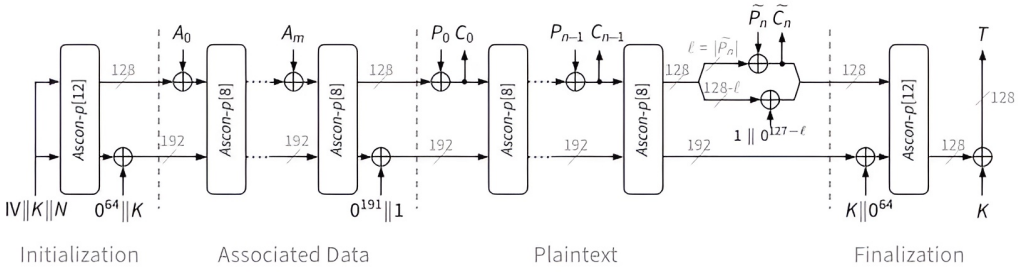


Figure 6.1: Ascon-AEAD128 encryption [33]

As specified in the previous section, the core of the algorithm is represented by *permutations*, denoted by p^a and p^b, which consist of a different number of *rounds* depending on the version of the algorithm used. In particular, for the ASCON-128a version used in this analysis, $a = 12$ and $b = 8$. Each round is composed by three phases acting on a 320-bit state, as shown in Figure 6.2:

- **Addition of round constant** (1): a 8-bit constant is added to the least significant byte of the state register S_2. Each round has its own round constant.
- **Substitution box (S-Box)** (2): the i-th column of the state is taken as input for the 5-bit S-Box. That same column is updated with the 5-bit S-Box output (b). The algebraic normal form of the S-Box is reported in Figure 6.1. Note that these expressions refer to operations performed on a single column of the state (bit-sliced form) and must be repeated on all 64 bits. Denoting

the state registers at the beginning of the round as $x_0, ..., x_4$ and the state at the output of the S-Box as $y_0, ..., y_4$, we have:

$$\begin{aligned}
y_0 &= x_4x_1 \oplus x_3 \oplus x_2x_1 \oplus x_2 \oplus x_1x_0 \oplus x_1 \oplus x_0 \\
y_1 &= x_4 \oplus x_3x_2 \oplus x_3x_1 \oplus x_3 \oplus x_2x_1 \oplus x_2 \oplus x_1 \oplus x_0 \\
y_2 &= x_4x_3 \oplus x_4 \oplus x_2 \oplus x_1 \oplus 1 \\
y_3 &= x_4x_0 \oplus x_4 \oplus x_3x_0 \oplus x_3 \oplus x_2 \oplus x_1 \oplus x_0 \\
y_4 &= x_4x_1 \oplus x_4 \oplus x_3 \oplus x_1x_0 \oplus x_1
\end{aligned} \tag{6.1}$$

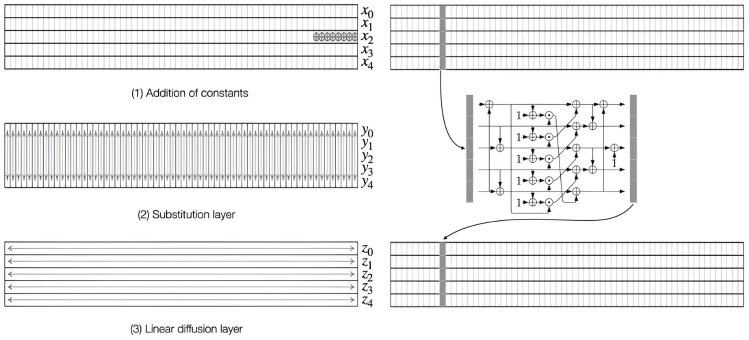
Since at the beginning of the initialization phase the initialization vector is stored in the register S_0, the two halves of the key in the registers S_1 and S_2 and the two halves of the nonce in the registers S_3 and S_4, the terms x_0, x_1, x_2, x_3, x_4 can be replaced with IV, k_0, k_1, n_0, n_1. The previous expressions can thus be rewritten as Equation 6.2:

$$\begin{aligned}
y_0 &= n_1k_0 \oplus n_0 \oplus k_1k_0 \oplus k_1 \oplus k_0IV \oplus k_0 \oplus IV \\
y_1 &= n_1 \oplus n_0k_1 \oplus n_0k_0 \oplus n_0 \oplus k_1k_0 \oplus k_1 \oplus k_0 \oplus IV \\
y_2 &= n_1n_0 \oplus n_1 \oplus k_1 \oplus k_0 \oplus 1 \\
y_3 &= n_1IV \oplus n_1 \oplus n_0IV \oplus n_0 \oplus k_1 \oplus k_0 \oplus IV \\
y_4 &= n_1k_0 \oplus n_1 \oplus n_0 \oplus k_0IV \oplus k_0
\end{aligned} \tag{6.2}$$

- **Linear diffusion layer** (3) The linear diffusion layer provides diffusion within each 64-bit word S_i. This layer applies the linear functions Σ_i to their corresponding state words as $S_i \leftarrow \Sigma_i(S_i)$, for $0 \leq i \leq 4$, where each Σ_i is defined as:

$$\begin{aligned}
\Sigma_0(S_0) &= S_0 \oplus (S_0 \ggg 19) \oplus (S_0 \ggg 28) \\
\Sigma_1(S_1) &= S_1 \oplus (S_1 \ggg 61) \oplus (S_1 \ggg 39) \\
\Sigma_2(S_2) &= S_2 \oplus (S_2 \ggg \ \ 1) \oplus (S_2 \ggg \ \ 6) \\
\Sigma_3(S_3) &= S_3 \oplus (S_3 \ggg 10) \oplus (S_3 \ggg 17) \\
\Sigma_4(S_4) &= S_4 \oplus (S_4 \ggg \ \ 7) \oplus (S_4 \ggg 41)
\end{aligned} \tag{6.3}$$

In practice, the linear shift layer performs a circular rotation of the bits in each state register. The contents of each state register are shifted to the right by a certain amount. The LSBs that would be lost in this process are instead moved to the left, occupying the positions that become free as a result of the shift operation.



(a) Three steps of a round (b) An S-box computation.

Figure 6.2: Display of the 3 steps of a single round [27]

The 320-bit state is divided into 5 words of 64 bits each (Equation 6.4). This allows mathematical expressions to be easily translated into the corresponding hardware or software implementation. In particular, each word can be stored in one or more registers, facilitating implementation on 8-bit, 32-bit, and 64-bit architectures.

$$S = S_0||S_1||S_2||S_3||S_4 \tag{6.4}$$

6.2 Sensitivity to side-channel attacks

Being aware of this vulnerability, ASCON designers took into account the threat posed by side-channel attacks as early as the design stage. For this reason, the 320-bit state is divided into 64-bit registers, which are updated after each operation. If an attacker wanted to perform a *Correlation Power Analysis* as done for AES, it would be necessary to calculate the correlation between the captured traces and all 2^{64} possible hypotheses for the secret key. Obviously, this makes the computation extremely demanding both in terms of the hardware resources and the computation time required on present-day computers.

Nevertheless, it is possible to perform CPA attacks on ASCON, as first described by *Samwel and Daemen* [35]. The vulnerable part of the process is the initialization phase. In this phase, the state is initially constructed by storing the 64 bits of the initialization vector in register S_0, the 128 bits of the key in registers S_1 and S_2, and the 128 bits of the nonce in registers S_3 and S_4.

The initialization phase is followed by the p^a permutation step, consisting of 12 rounds. Of these, the first round is the target of the side-channel attack, since at this point the initial state content is known and the key is used for the first time.

Since each round consists of three steps, various points can be targeted by a side channel attack, such as the output value of the S-Box, as was also the case

with AES, and the result of the linear shift layer. Of the two, the latter proves to be effective for the attack, as reported by [27]. In their article, the researchers analyse the reasons why the attack is ineffective when the S-Box output is used as a selection function, concluding that this model does not guarantee uniqueness in the correlation of the results obtained. For this reason, the output of the linear shift layer is preferred and is used as the selection function for this analysis.

More in detail only the expression for y_0^j, y_1^j, y_4^j from Equation 6.2 can be considered for the attack as their computations contain non-linear terms between the key and the nonce, as explained by [35]. For the sake of clarity, the expression for y_0^j is shown below:

$$y_0^j = n_1^j k_0^j \oplus n_0^j \oplus k_1^j k_0^j \oplus k_1^j \oplus k_0^j IV^j \oplus k_0^j \oplus IV^j \tag{6.5}$$

A useful simplification can be made to this expression. In fact, assuming that the key is always the same during the analysis, the quantity $k_1^j k_0^j \oplus k_1^j \oplus k_0^j IV^j \oplus IV^j$ can be removed since it contains only constant terms that do not depend on the value of the nonce [36]. Therefore, these will contribute a constant amount to the power consumption, which is then eliminated by performing a differential analysis. The simplified expression for the S-Box output becomes:

$$\tilde{y}_0^j = k_0^j(n_1^j \oplus 1) \oplus n_0^j \tag{6.6}$$

Since the target of the attack is the output of the linear shift layer, this operation is taken into account on the newly calculated quantity. From Equation 6.3, the output z_0 for the state register S_0 is:

$$z_0 = y_0 \oplus (y_0 \ggg 19) \oplus (y_0 \ggg 28) \tag{6.7}$$

The i-th bit of z_0 $(0 \leq i \leq 63)$ thus can be computed as:

$$z_0^j = y_0^j \oplus y_0^{j+19} \oplus y_0^{j+28} \tag{6.8}$$

or, more explicitly

$$\begin{aligned} \tilde{z}_0^j = {} & (k_0^j(n_1^j \oplus 1) \oplus n_0^j) \\ & \oplus (k_0^{j+36}(n_1^{j+36} \oplus 1) \oplus n_0^{j+36}) \\ & \oplus (k_0^{j+45}(n_1^{j+45} \oplus 1) \oplus n_0^{j+45}) \end{aligned} \tag{6.9}$$

As can be observed, the sole unknowns in the Expression 6.9 are the 3 bits $k_0^j, k_0^{j+19}, k_0^{j+28}$ of the most significant half of key. From each bit of the state register S_0 at the output of the linear shift layer, it is therefore possible to recover 3 bits of the key. This offers the possibility of performing a correlation power analysis on the 2^3 possible combinations of these 3 bits.

In a similar fashion, it is possible to derive the expression for the state register S_1, as reported in Equation 6.10:

$$\begin{aligned} \tilde{z}_1^j &= (n_0^j(k_{01}^j \oplus 1) \oplus n_1^j) \\ &\oplus (n_0^{j+61}(k_{01}^{j+61} \oplus 1) \oplus n_1^{j+61}) \\ &\oplus (n_0^{j+39}(k_{01}^{j+39} \oplus 1) \oplus n_1^{j+39}) \end{aligned} \tag{6.10}$$

where $k_{01}^j = k_0^j \oplus k_1^j$. As opposed to the most significant half of the key k_0, the bits of the half relating to k_1 are recovered indirectly using the bits recovered by attacking $\tilde{z}_0^j$.

6.3 Measurement setup

The experiment setup for the **CPA** attack on ASCON's software implementation is shown in Figure 6.3. The *X-HEEP* microcontroller is synthesised on the FPGA of the *CW305* board and operates at a frequency of 10 MHz.

As explained in the previous section, the linear shift layer is chosen as the attack point, and Hamming weight is used as leakage model for the state registers S_0 and S_1. As for these two words, their content is divided between two CPU registers each, since the microcontroller used for this analysis has a 32-bit architecture.

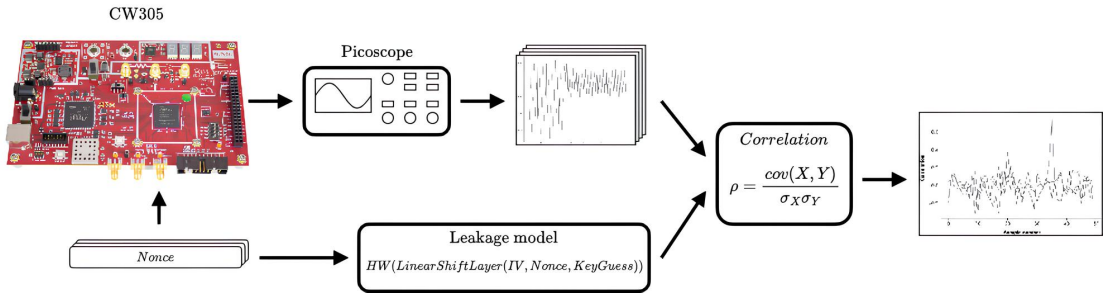


Figure 6.3: Measurement setup for the CPA attack on ASCON

In order to reduce the number of samples per trace, the oscilloscope is triggered only at the beginning of the linear shift layer, and the observation time covers the two instructions that update the above-mentioned registers, as show in the code snippet below. With this approach, each trace contains 1125 samples.

```
1 s->x[0] ^=
2     (s->x[0] >> 19) ^ (s->x[0] << 45) ^ (s->x[0] >> 28) ^ (s->x[0] << 36);
3 s->x[1] ^=
4     (s->x[1] >> 61) ^ (s->x[1] << 3) ^ (s->x[1] >> 39) ^ (s->x[1] << 25);
```

The algorithm has been modified to implement the mechanism for triggering the oscilloscope, as discussed in detail in the section on AES. The same configuration was also used for the oscilloscope.

The Figure 6.4 shows the plot of the measured power traces. It can be seen that the absorption peaks are concentrated in only two groups, each corresponding to the update of the two state words S_0 and S_1.

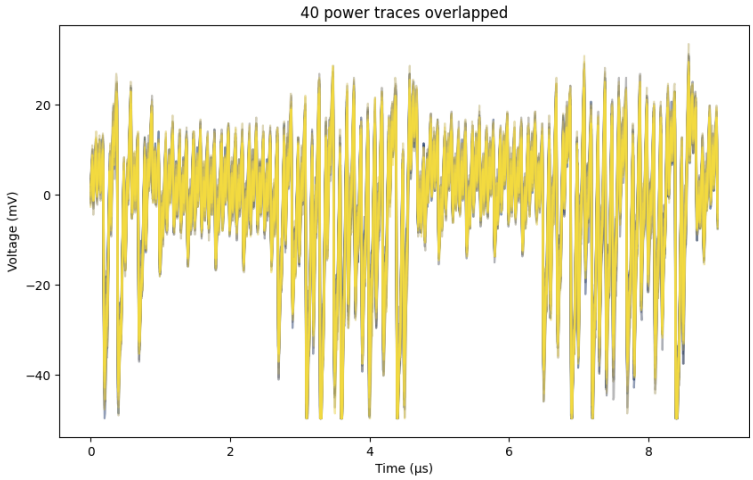


Figure 6.4: Plot of 40 captured power traces

6.4 CPA attack

As already done for AES, the objective of this analysis is to demonstrate the impact of the S-Box used on the sensitivity of the algorithm to side-channel attacks. In this regard, the selection function used is not the one reported in Equation 6.9 and Equation 6.10, but a more generic model that takes into account all the operations performed in a single round, namely: round constant addition, substitution layer and linear shift layer. In this way, the results predicted by the model always reflect the operations actually performed by the device. It is only necessary to specify the S-Box used.

More in detail, the model is built as follows:

- The $j, j + c_0, j + c_1$ bits are extracted from the *initialization vector* IV, and the two *nonce* halves n_0 and n_1. The shift amount value c_j is defined by the index of the attacked state register, as reported in Equation 6.3. For instance, if the attacked register is S_0, the two shift amount values c_0 and c_1 will be 19 and 28, respectively.

- A key guess of 3 bits is chosen for both the two halves of the key k_0 and k_1. The the indexes of the 3 bits of each key guess follow the same rule $j, j + c_0, j + c_1$.

- The 5-bit input of the S-Box is built from these values, in the following order (the most significant bit is represented by IV_j):

$$\begin{aligned} input_j &= (IV^j, k_0^j, k_1^j, n_0^j, n_1^j) \\ input_{j+c_0} &= (IV^{j+c_0}, k_0^{j+c_0}, k_1^{j+c_0}, n_0^{j+c_0}, n_1^{j+c_0}) \\ input_{j+c_1} &= (IV^{j+c_1}, k_0^{j+c_1}, k_1^{j+c_1}, n_0^{j+c_1}, n_1^{j+c_1}) \end{aligned} \tag{6.11}$$

- The 5-bit output of the S-Box is computed for each of the previous inputs

$$\begin{aligned} output_j &= SBox(input_j) \\ output_{j+c_0} &= SBox(iinput_{j+c_0}) \\ output_{j+c_1} &= SBox(input_{j+c_1}) \end{aligned} \tag{6.12}$$

- The XOR operation defined by the linear shift layer is applied to the 3 outputs of the S-Box

$$Z = (output_j) \oplus (output_{j+c_0}) \oplus (output_{j+c_1}) \tag{6.13}$$

- The bit corresponding to the attacked state register is extracted. The Hamming Weight of that bit is used as leakage model (since there is just 1 bit, the Hamming Weight coincides with the bit value itself).

For greater clarity, Figure 6.5 shows a diagram of the process described above when the attacked register is S_0.

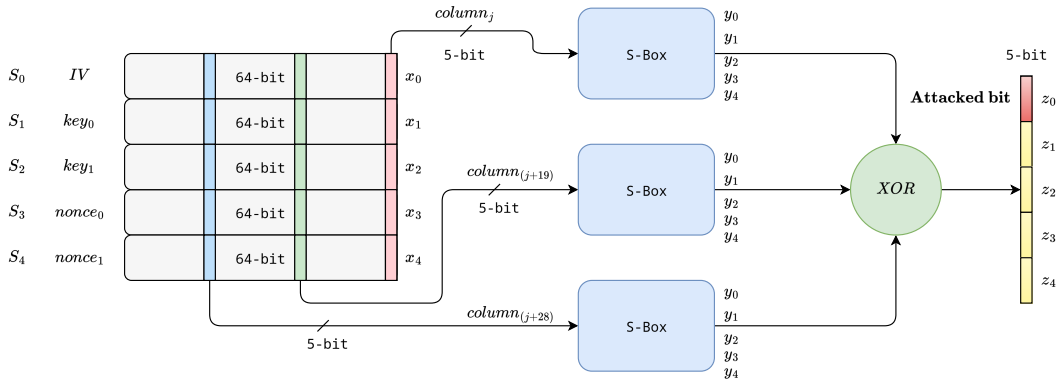


Figure 6.5: Leakage model used for the analysis on ASCON.

In the case of the standard Ascon S-Box, the model is reduced to Equation 6.9 and Equation 6.10, with the assumption that the 3 bits of the key guess for k_1 are all equal to 0. This assumption proved to be correct, since the 64 results of the CPA attack on the S_0 state register had the same correlation value in groups of 8, confirming that the hypothesis made on the 3 bits of k_1 is not significant for the attack on that register.

The S-Boxes used for this analysis are listed in Table 6.1. Their cryptographic properties and a summary table of the results obtained from the CPA attack are provided in the Table 6.2 and Table 6.3. Chapter 7 will explore the meaning of the metrics shown in the table in more detail, considering both resistance to classical cryptanalysis and resistance to side-channel attacks. A different number of traces were collected for each of them.

S-Box	ASCON	Bilgin	Allouzi	LU 4	LU 5	LU 6	LU 7
Number of traces	10k	150k	150k	150k	150k	150k	150k

Table 6.1: Tested S-Boxes for the analysis on ASCON with the number of collected traces

SBox	**Power analysis resistance**				**Cryptanalysis resistance**		
	Ascon HW		**Ascon SW**		**Non**	**Differential**	**Confusion**
	Traces	Key bits	Traces	Key bits	**linearity**	**Uniformity**	**Coefficient**
Ascon	60k	128/128	10k	128/128	8	8/32	0.5016
Bilgin [37]	1M	80/128	150k	98/128	12	2/32	0.3080
Allouzi [38]	1M	88/128	150k	95/128	12	2/32	0.4048
Lu 4 [39]	60k	96/128	150k	86/128	8	8/32	0.5824
Lu 5 [39]	1M	110/128	150k	107/128	8	8/32	0.2233
Lu 6 [39]	60k	86/128	150k	81/128	8	8/32	0.8887
Lu 7 [39]	60k	97/128	150k	81/128	8	6/32	0.7072

Table 6.2: Trade-off between power analysis resistance and cryptanalytic properties for ASCON S-Box

The results of the analysis showed that fewer than 10,000 traces are necessary to recover the entire key from ASCON's software implementation with the standard S-Box. These results are consistent with those reported by *Picek*'s analysis on the ASCON's software implementation on a STM32 microcontroller, which reported

Sbox type	NL min	NL max	DU	CCV	MCC	TO
Ascon	8	12	8	0.5016	0.2500	4.2581
Lut Bilgin	12	12	2	0.3080	0.3750	4.8387
Lut Shamash	12	12	2	0.4048	0.3750	4.8387
Lut Lu 4	8	12	8	0.5824	0.1719	4.4798
Lut Lu 5	8	12	8	0.2233	0.3750	4.4839
Lut Lu 6	8	12	8	0.8887	0.2500	4.3871
Lut Lu 7	8	12	6	0.7072	0.3438	4.4032

Table 6.3: Cryptographic properties for the S-boxes in exam for ASCON. NL: nonlinearity. DU: differential uniformity. CCV: confusion coefficient variance. MCC: minimum confusion coefficient. TO: transparency order

that approximately 8,000 traces are sufficient to recover the entire key [40].

With regard to the choice of which bits to attack, the researchers [27] provide a table of suggested indices computed with the aim of reducing the total number of bits to be attacked. Since 3 bits of the key can be recovered from each of the states register bits, it is not necessary to attack all 64 bits of each register, as many of the recovered bits would be overwritten multiple times. In their analysis, the researchers report that they performed 23 CPA attacks on register S_0 and 24 on register S_1.

In this analysis, however, the signal-to-noise ratio was considered as the criterion of choice for the indices of the bits to be attacked. In particular, the attacks were performed on the bits with a higher SNR, proceeding in descending order until all the bits in the range 0-63 had been recovered. This approach required a greater number of CPA attacks overall (34 for the register S_0 and 37 for the register S_1), but on the other hand, far fewer traces were needed to recover the entire key.

The list of the indexes of the attacked bits is reported in Table 6.4.

State register	**Attacked bits**
S_0	32 13 34 4 6 54 36 0 33 63 7 16 55 19 17 41 1 40 8 48 24 39 14 31 58 49 56 47 37 29 15 46 57 11
S_1	32 0 63 1 14 13 36 15 31 8 38 43 5 18 23 12 45 16 9 42 3 51 2 49 24 20 44 40 28 30 37 19 47 59 53 4 46

Table 6.4: ASCON standard S-Box: attacked bits, ordered according to their SNR

The signal-to-noise ratio appears to be a promising method for selecting which bits to attack. Figure 6.6 and Figure 6.7 show the comparison between bit 33, which has a higher SNR, and bit 48. It can be seen that for the attack on bit 33, the correlation value obtained for the correct key guess is much higher for the same number of traces. Similarly, for the same correlation value, a much smaller number of traces is required.

The Figure 6.8 and Figure 6.9 show the SNR graph plotted over the measured power traces for both bits. It is interesting to note that the maximum value for the signal-to-noise ratio occurs at a peak power absorption for both cases, further indicating that the S_0 status register is updated at that point.

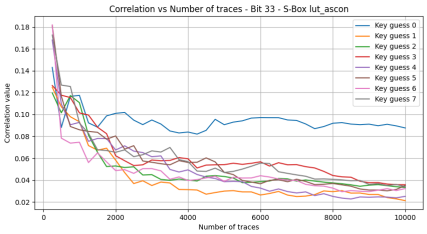


Figure 6.6: Correlation values for the bit 33

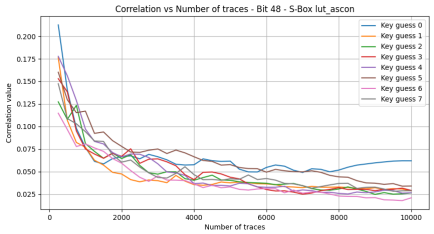


Figure 6.7: Correlation values for the bit 48

Figure 6.8: SNR value for the bit 33

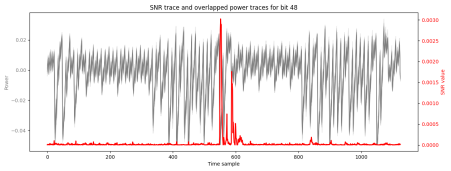


Figure 6.9: SNR value for the bit 48

For the analysis on different S-Boxes, 150,000 power traces were collected for each of them. The procedure followed for the attack was the same used for the standard S-Box:

- SNR computation
- Definition of bits to attack
- CPA attack
- Evaluation of results

The results show that the S-Boxes analysed behave differently, some better than others, but in no case was it possible to recover all the key bits. An important

detail is that even for some of the bits with a high SNR, the correlation value of the correct key guess is lower than that of other incorrect key guesses. As a result, the recovered key bits are different from what was expected.

Figure 6.10 and Figure 6.11 show the correlation value trend for all key guesses as the number of traces for bit 48 increases. It can be seen that for the *bilgin* S-Box, the correlation of the correct key guess remains similar to that of the incorrect key guesses, even though the SNR value of that bit calculated on the data measured for the *bilgin* S-Box is still greater than the SNR value for the same bit measured on the standard S-Box data.

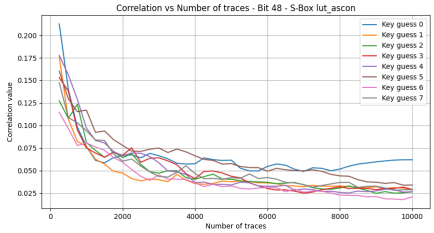


Figure 6.10: Correlation plot as function of traces for bit 48 - Standard S-Box. SNR = 0.003856

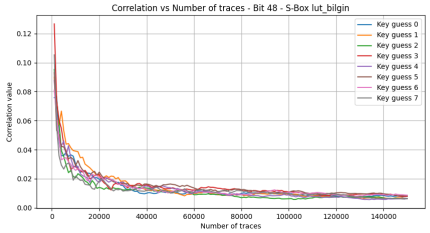


Figure 6.11: Correlation plot as function of traces for bit 48 - Bilgin S-Box. SRN = 0.007813

6.5 Considerations

The results obtained clearly show that the type of S-Box used influences the effectiveness of power side-channel attacks against the ASCON algorithm. In particular, using the standard S-Box, it was possible to recover the entire key with less than 10,000 traces, while for all other S-Boxes, 150,000 traces were not sufficient.

It is interesting to note that there is an order of magnitude difference between the two cases. This shows that the use of different S-Boxes represents a possible lightweight countermeasure against this type of attack, with minimal additional overhead.

Unlike the case of AES, where the effect of different types of S-Boxes was more limited, in the case of ASCON the advantages are much more pronounced, probably due to the very nature of the two algorithms. In fact, the strength of side-channel attacks lies in their ability to exploit information leaked from the device into the surrounding environment. In the case of AES, the key is used in 1-byte blocks. This makes it relatively easy to implement a CPA attack as the assumption is made on 1 byte of information. In the case of ASCON, however, the CPA attack is performed on a single bit, which inevitably presents less information that can be exploited for the attack. For this reason, the characteristics of the S-Box used

seem to have a more significant impact on sensitivity to this type of attack.

On the other hand, this type of countermeasure introduces a significant trade-off. The use of S-Boxes with different mathematical characteristics from the default S-Box reduces the algorithm's resistance to standard cryptanalysis, so it is necessary to find a balance between the two properties.

Chapter 7

S-Box metrics: State of the Art

In recent years, the threat posed by side channel attacks has received increasing attention from the research community due to their particular effectiveness compared to classical cryptography attacks. In fact, an attacker will choose the type of attack based on the complexity of implementation, favouring by far the simplest one. In the context of embedded systems, the choice will always fall on side-channel attacks, as the difference in the amount of data needed to break the algorithm is many orders of magnitude [41].

For this reason, one of the main focuses of this thesis was to verify the effectiveness of different S-Boxes as a lightweight countermeasure against power side-channel attacks for the well-known `AES` algorithm and for the new `ASCON` lightweight cryptography standard. At this point, one might wonder why the analysis focused on this type of countermeasure and not on one of the other types listed in Section 4.5. After all, some of those countermeasures have proven to be particularly effective, such as the masking technique [42].

The main reason why the S-Box was chosen as the countermeasure essentially concerns the implementation overhead associated with the countermeasures themselves. The use of masking introduces a significant performance overhead, making its use impractical on embedded devices where both computing power and memory size are extremely limited and the chip area must fall within certain limits.

Based on this severe penalty, some researchers have suggested considering resistance to side-channel attacks as early as the algorithm design stage, on a par with all other parameters used to define resistance against standard cryptanalysis. This approach would make the addition of further countermeasures unnecessary and avoid the overhead associated with them [43].

As many of the properties that make an algorithm resistant to classical cryptanalysis depend on the design of the S-Box and since the S-Box is the main target of many side-channel attacks (as in the case of AES), some researchers suggest incorporating resistance parameters against side-channel attacks into the S-Box itself [44].

In this way, the S-Box becomes a countermeasure against side-channel attacks by its very nature, without introducing any additional overhead or significant differences in the required hardware resources. However, there are some caveats to consider when constructing the S-Box.

For an encryption algorithm to be considered secure, it must satisfy the criteria of confusion and diffusion. The S-Box's role is to introduce confusion through a non-linear transformation of the input bits. While this property ensures the algorithm is secure against classical cryptanalysis, it is also a primary source of information that can be exploited by side-channel analysis. In fact, the use of a non-linear function inevitably involves the toggle of many of the output bits, and this inevitably translates into higher power consumption, for the reasons explained in Section 4.

In the construction of S-Boxes, there is therefore a trade-off between resistance to classical cryptographic analysis and side-channel analysis [44]. The following sections list some of the properties used to define the quality of an S-Box depending on the type of analysis considered.

7.1 Classical cryptanalysis resistance metrics

When it comes to classical cryptanalysis, the most important parameters defining the quality of an S-Box certainly include *non-linearity* and *differential uniformity*, which are defined as follows [45]:

- **Non-linearity**[44]: The term S-Box stands for *Substitution box*. Its role is to apply a transformation to the input bits using a specific function f. This function must be chosen so as to resist *linear cryptanalysis*, which is why the non-linearity parameter is introduced. Non-linearity is defined as the minimum *Hamming Distance* between function f and *affine functions*, *i.e.* all those functions that can be expressed in the form represented by Equation 7.1, where x represents the n-bit input vector, A is a binary matrix of size $m \times n$, while b is a constant vector of m-bits.

$$f(x) = \mathbf{A}x \oplus b \tag{7.1}$$

  High non-linearity is desirable because it ensures resistance to linear cryptanalysis.

- **Differential Uniformity**[44]: is a measure of the `S-Box` resistance to *differential cryptanalysis*. It is defined as the maximum number of times any specific output difference can result from a given input difference, across all possible input pairs. In other words, this quantity represents the probability that a difference in the input data of the `S-Box` will lead to a difference in the output of the `S-Box`. Functions f that have differential uniformity equal to 2 are called the Almost Perfect Nonlinear (APN) functions. A differential uniformity value close to 2 ensures resistance to differential cryptanalysis.

7.2 Side channel attack resistance metrics

Over the years, studies have been conducted on classical cryptanalysis resistance metrics to assess their impact on effectiveness against side-channel attacks. In particular, it has been shown that metrics that make an `S-Box` secure against classical cryptanalysis conflict with security against side-channel attacks, as demonstrated by Prouff [43]. Therefore, it is often necessary to find a compromise.

For example, a high non-linearity value is desirable as it makes the `S-Box` resistant to linear cryptanalysis, but at the same time, it makes it very sensitive to side-channel attacks. A high non-linearity value implies a large Hamming Distance between the input value of the `S-Box` and the corresponding output. This Hamming Distance inevitably translates into a large variation in power consumption that can be easily measured and exploited by an attacker. Intuitively, to make an S-Box secure against side-channel attacks, the Hamming Distance between an input variable x and the output value `S-Box`(x) must be minimal, so as to cause a reduced variation in power consumption. However, this property would make it extremely vulnerable to classical cryptanalysis attacks.

On the contrary, if only one bit of the `S-Box` output is considered, the greater the resistance to side-channel attacks, the lower the resistance to differential cryptanalysis, although, as proven by Carlet [44], this statement is not necessarily true when considering the entire output of the `S-Box`.

It is therefore clear that new metrics need to be introduced to evaluate the resistance of `S-Box`es to side-channel attacks, as summarized in [46]:

- **Confusion Coefficient Variance (CCV)**: Before defining this metric, it is worth exploring the concept of *Confusion Coefficient*, introduced by Fei [47]. This metric measurers the probability of occurrences for which key hypotheses k_i and k_i result in different intermediate values ν. When performing a DPA attack, this parameter can be computed according to the Equation 7.2, where $\mathfrak{L}$ denotes the leakage function, p denotes the arbitrary inputs, and $\mathbb{E}$ is the mean operator:

$$k(k_i, k_j) = \mathbb{E}\left[(\mathfrak{L}(F(k_i \oplus p)) - \mathfrak{L}(F(k_j \oplus p)))^2\right] \tag{7.2}$$

Some years later Picek [48], proposed to calculate the variance of all confusion coefficients with respect to each possible k_i and k_j under the Hamming Weight leakage model. He proved that `S-Box` with higher *confusion coefficient variance* (CCV) value leads to a higher resistance against side-channel attacks. This metric can be computed as shown in Equation 7.3.

$$CCV(F) = var\left(\mathbb{E}\left[(H(F(k_i \oplus p)) - H(F(k_j \oplus p)))^2\right]\right) \tag{7.3}$$

- **Minimum Confusion Coefficient (MCC)**: In their analysis, Guilley [49] et al. emphasise that, in the context of DPA or CPA attacks and low SNR, S-boxes with the lowest confusion coefficient are the most resistant to side-channel attacks. The lower the value of MCC, the lower the success probability to extract the secret key based on leakages associated with the S-Box. MCC can be computed as shown in Equation 7.4.

$$MCC(F) = \min_{k \neq k^*}\left(\mathbb{E}\left\{\left(\frac{\mathfrak{L}(F(k^* \oplus p)) - \mathfrak{L}(F(k \oplus p))}{2}\right)^2\right\}\right) \tag{7.4}$$

- **Transparency Order (TO)**: This metric was formally defined by Prouff [43] and indicates the degree of resistance of an `S-Box` against DPA attacks. The lower the *transparency order* value of the `S-Box`, the greater its resistance to DPA attacks. In his article, the author also reports expressions for calculating an upper bound and a lower bound for the *TO*. The latter in particular represents the maximum resistance value against DPA attacks that can be achieved by an `S-Box`. The TO of an (n, m) `S-Box` is defined as the Equation 7.5, where D_aF is the derivative of F with respect to a vector a, and W_{D_aF} is the Fourier transform of D_aF.

$$\begin{aligned} TO(F) = max_{\beta in F_2^m}(|m - 2H(\beta)| - \\ \frac{1}{2^{2n} - 2^n} \sum_{a \in F_2^{n*}} | \sum_{v \in F_2^m, H(v)=1} (-1)^{v \cdot \beta} W_{D_aF}(0, v)|) \end{aligned} \tag{7.5}$$

Now that these new metrics have been introduced, it is appropriate to verify whether the theoretical results of `S-Box`es constructed according to these resistance parameters reflect reality, highlighting effective resistance to side-channel attacks.

A theoretical analysis conducted by Li et al. [46] on 11 S-boxes of different sizes, more precisely 4×4 and 8×8, demonstrated the existence of an effective

relation between the *improved transparency order* VTO (a slightly modified version of the TO) and the *confusion coefficient variance* CCV. In particular, the trend of these two metrics is consistent: lower VTO and higher CCV values correspond to S-Boxes that are more resistant to side-channel attacks. On the other hand, the ordering of S-Boxes obtained using the *minimum confusion coefficient* MCC as a metric is not consistent with either VTO or CCV.

The authors also point out that larger S-Boxes lead to significantly higher values of VTO and MCC, which implies S-Boxes with larger sizes are more vulnerable against SCAs.

In the second part of their analysis, the authors focus on the experimental results obtained by performing a CPA attack on all the S-Boxes analysed theoretically. The results clearly demonstrated the connection between VTO and CCV, revealing that S-Boxes with lower VTO values and higher CCV values are more resistant to CPA. However, when the difference of the VTO (or CCV) values of the two S-Boxes is relatively small, these two metrics lack the accuracy to order the S-Boxes according to their resiliency, resulting in inconsistencies with the theoretical results.

According to the authors, this inconsistency is due to the different ways in which the leakage model is used in constructing the metrics and in the CPA attack. In the former case, the Hamming Weight of an S-Box output is used as the leakage model, whereas in the latter case, the Pearson's correlation coefficient is used to compute a relationship between the traces and the Hamming Weight of the S-Box output.

7.2.1 AES SoA S-Boxes

With regard to the analysis conducted in this thesis, the experimental results obtained using the 8×8 S-Boxes in the AES case actually demonstrate behaviour consistent with Li's statement [46]. The analysis revealed that the $Freyre_2$ S-Box offered the greatest resistance to CPA attacks, followed by the $Freyre_1$ S-Box. Both S-Boxes have the lowest TO values and the highest CCV values. $Freyre_1$ should theoretically perform better, but the difference between the TO and CCV values of the two S-Boxes is so small that the same inconsistency between the theoretical model and the experimental results highlighted by Li also appears in this case.

Nevertheless, the advantage of using SCA-resilient S-Boxes is not particularly significant in the context of AES.

7.2.2 ASCON SoA S-Boxes

In the case of ASCON, however, the two 5×5 S-Boxes that demonstrated the greatest resistance to CPA attacks were *Lut* LU_6 and *Lut* LU_7, resulting in the

lowest success rates. According to the theoretical model, these two `S-Box`es have the highest CCV values. The idea that the higher the CCV value, the higher the SCA resistance appears to be true this time. However, in this case, the difference in TO values is less pronounced. These two `S-Box`es show slightly higher TO values than the standard `ASCON S-Box`; theoretically, they should therefore perform worse. However, the difference is very small, and the situation could fall within the inconsistency issue identified by Li.

Despite these inconsistencies, the CPA attack on all the modified `S-Box`es proved ineffective in recovering all the key bits with 150,000 traces. This clearly demonstrates that choosing a different `S-Box`es can be an effective countermeasure against SCA when addressing the ASCON algorithm.

Chapter 8

Conclusion

This thesis explored the vulnerability of cryptographic algorithm implementations to power side-channel attacks, with a primary focus on the well-known *Advanced Encryption Standard* (AES) and the new ASCON family of lightweight cryptographic algorithms.

Analysis was conducted on a microcontroller based on RISC-V architecture, synthesised on a *ChipWhisperer* board. It was demonstrated that low-cost side-channel analysis can be conducted using only open-source tools. Furthermore, the analysis revealed that even this type of platform is susceptible to side-channel attacks as it exhibits measurable leakage that could compromise security.

A hardware module called Bridge2Xheep was developed to enable communication between the host PC and the microcontroller, and to allow instructions to be loaded into the microcontroller's RAM during the boot phase.

After testing the correct functioning of the platform, side channel analysis was performed by executing CPA attacks on AES and ASCON. In particular, the effectiveness of appropriately designed S-boxes in improving the resistance of algorithms against side-channel attacks was analysed. For the ASCON case, the selection of the bits to attack was made using the *signal-to-noise ratio* (SNR) as a metric. This metric proved effective, as attacks on bits with a high SNR were successful, unlike those on bits with a low SNR.

The analysis showed that the choice of S-Box impacts the success of the attack for both AES and ASCON and can therefore be seen as a lightweight countermeasure.

Finally, an investigation was conducted into the metrics used to evaluate the resistance of S-boxes against classical cryptanalysis and side-channel analysis. The

validity of these metrics was then compared with the obtained results, demonstrating that the transparency order (or rather, the improved transparency order) and the confusion coefficient variance are useful metrics for constructing `S-box`es that are resistant to side-channel attacks.

Bibliography

[1] Jean-Philippe Aumasson. *Serious cryptography: a practical introduction to modern encryption*. No Starch Press, Inc, 2024 (cit. on p. 1).

[2] Suetonius. *Vita Divi Julii*. "56.6". URL: `https://thelatinlibrary.com/suetonius/suet.caesar.html#56` (cit. on p. 1).

[3] Hans Delfs, Helmut Knebl, and Helmut Knebl. *Introduction to cryptography*. Vol. 2. Springer, 2002 (cit. on p. 2).

[4] Ronald L Rivest, Adi Shamir, and Leonard Adleman. «A method for obtaining digital signatures and public-key cryptosystems». In: *Communications of the ACM* 21.2 (1978), pp. 120–126 (cit. on p. 2).

[5] Jan Pelzl ChristofPaar and Bart Preneel. «Understanding Cryptography: A Textbook for Students and ractitioners». In: *Springer* (2010) (cit. on p. 2).

[6] YongBin Zhou and DengGuo Feng. «Side-channel attacks: Ten years after its publication and the impacts on cryptographic module security testing». In: *Cryptology ePrint Archive* (2005) (cit. on pp. 3, 6).

[7] Paul C Kocher. «Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems». In: *Advances in Cryptology—CRYPTO'96: 16th Annual International Cryptology Conference Santa Barbara, California, USA August 18–22, 1996 Proceedings 16*. Springer. 1996, pp. 104–113 (cit. on p. 3).

[8] Paul Kocher, Joshua Jaffe, and Benjamin Jun. «Differential power analysis». In: *Advances in Cryptology—CRYPTO'99: 19th Annual International Cryptology Conference Santa Barbara, California, USA, August 15–19, 1999 Proceedings 19*. Springer. 1999, pp. 388–397 (cit. on pp. 3, 35, 37).

[9] Paul Kocher, Ruby Lee, Gary McGraw, and Anand Raghunathan. «Security as a new dimension in embedded system design». In: *Proceedings of the 41st annual design automation conference*. 2004, pp. 753–760 (cit. on p. 3).

[10] YongBin Zhou and DengGuo Feng. «Side-channel attacks: Ten years after its publication and the impacts on cryptographic module security testing». In: *Cryptology ePrint Archive* (2005) (cit. on p. 3).

[11] P. Wright. *Spycatcher: The Candid Autobiography of Senior Intelligence Officer*. Stoddart, 1987. ISBN: 9780773721685. URL: https://books.google.it/books?id=grQ_HMEsmG4C (cit. on p. 4).

[12] Dmitri Asonov and Rakesh Agrawal. «Keyboard acoustic emanations». In: *IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004*. IEEE. 2004, pp. 3–11 (cit. on p. 4).

[13] Moritz Lipp et al. «Meltdown». In: *arXiv preprint arXiv:1801.01207* (2018) (cit. on p. 4).

[14] Paul Kocher et al. «Spectre attacks: Exploiting speculative execution». In: *Communications of the ACM* 63.7 (2020), pp. 93–101 (cit. on p. 4).

[15] Elke De Mulder, Samatha Gummalla, and Michael Hutter. «Protecting RISC-V against side-channel attacks». In: *Proceedings of the 56th Annual Design Automation Conference 2019*. 2019, pp. 1–4 (cit. on p. 6).

[16] Hao Cheng, Daniel Page, and Weijia Wang. «eLIMInate: a Leakage-focused ISE for Masked Implementation». In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2024.2 (2024), pp. 329–358 (cit. on p. 6).

[17] Qi Tian, Hao Cheng, Chun Guo, Daniel Page, Meiqin Wang, and Weijia Wang. «A Code-Based ISE to Protect Boolean Masking in Software». In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2025.2 (2025), pp. 293–332 (cit. on p. 6).

[18] Alessandro Barenghi, Luca Breveglieri, Niccolo Izzo, and Gerardo Pelosi. «Exploring cortex-M microarchitectural side channel information leakage». In: *IEEE Access* 9 (2021), pp. 156507–156527 (cit. on p. 6).

[19] Simone Machetti, Pasquale Davide Schiavone, Thomas Christoph Müller, Miguel Peón-Quirós, and David Atienza. *X-HEEP: An Open-Source, Configurable and Extendible RISC-V Microcontroller for the Exploration of Ultra-Low-Power Edge Accelerators*. 2024. arXiv: 2401.05548 [cs.AR] (cit. on pp. 7, 17).

[20] Colin O'flynn and Zhizhang Chen. «Chipwhisperer: An open-source platform for hardware embedded security research». In: *Constructive Side-Channel Analysis and Secure Design: 5th International Workshop, COSADE 2014, Paris, France, April 13-15, 2014. Revised Selected Papers 5*. Springer. 2014, pp. 243–260 (cit. on p. 8).

[21] Krste Asanović and David A Patterson. «Instruction sets should be free: The case for risc-v». In: *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2014-146* (2014) (cit. on p. 13).

[22] Andrew Waterman, Yunsup Lee, David A Patterson, and Krste Asanovic. «The RISC-V instruction set manual, volume I: User-level ISA, version 2.0». In: *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2014-54* (2014), p. 4 (cit. on p. 13).

[23] Wilson Snyder. «Verilator and systemperl». In: *North American SystemC Users' Group, Design Automation Conference*. Vol. 79. 2004, pp. 122–148 (cit. on p. 23).

[24] Olof Kindgren. «A scalable approach to IP management with FuseSoC». In: *1st Workshop on Open-Source Design Automation (OSDA)*. Vol. 5. 2019 (cit. on p. 30).

[25] Jasper Van Woudenberg and Colin O'Flynn. *The hardware hacking handbook: breaking embedded security with hardware attacks*. No Starch Press, 2021 (cit. on p. 35).

[26] Eric Brier, Christophe Clavier, and Francis Olivier. «Correlation power analysis with a leakage model». In: *Cryptographic Hardware and Embedded Systems-CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings 6*. Springer. 2004, pp. 16–29 (cit. on pp. 39, 40).

[27] Viet Sang Nguyen, Vincent Grosso, and Pierre-Louis Cayrel. «Practical Second-Order CPA Attack on Ascon with Proper Selection Function». In: (), p. 4 (cit. on pp. 41, 57, 58, 63).

[28] Emmanuel Prouff and Matthieu Rivain. «Masking against side-channel attacks: A formal security proof». In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2013, pp. 142–159 (cit. on p. 44).

[29] Kris Tiri and Ingrid Verbauwhede. «A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation». In: *Proceedings Design, Automation and Test in Europe Conference and Exhibition*. Vol. 1. IEEE. 2004, pp. 246–251 (cit. on p. 44).

[30] Joan Daemen and Vincent Rijmen. *The design of Rijndael: AES — the Advanced Encryption Standard*. Springer-Verlag, 2002, p. 238. ISBN: 3-540-42580-2 (cit. on p. 45).

[31] *Specification for the Advanced Encryption Standard (AES)*. Federal Information Processing Standards Publication 197. 2001. URL: `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf` (cit. on p. 45).

[32] Muhammad Mushtaq, Sapiee Jamel, Abdulkadir Disina, Zahraddeen Pindar, Nur Shakir, and Mustafa Mat Deris. «A Survey on the Cryptographic Encryption Algorithms». In: *International Journal of Advanced Computer Science and Applications* 8 (Nov. 2017), pp. 333–344. DOI: 10.14569/IJACSA.2017.081141 (cit. on p. 46).

[33] Meltem Sönmez Turan, Kerry McKay, Donghoon Chang, Jinkeon Kang, and John Kelsey. *Ascon-based lightweight cryptography standards for constrained devices: authenticated encryption, hash, and extendable output functions*. Tech. rep. National Institute of Standards and Technology, 2024 (cit. on pp. 54, 55).

[34] Chad Boutin. «NIST Selects 'Lightweight Cryptography'Algorithms to Protect Small Devices». In: *US Department of Commerce, National Institute of Standards and Technology, Tech. Rep.* (2023) (cit. on p. 54).

[35] Niels Samwel and Joan Daemen. «DPA on hardware implementations of Ascon and Keyak». In: *Proceedings of the Computing Frontiers conference*. 2017, pp. 415–424 (cit. on pp. 57, 58).

[36] Guido Bertoni, Joan Daemen, Nicolas Debande, Thanh-Ha Le, Michael Peeters, and Gilles Van Assche. «Power analysis of hardware implementations protected with secret sharing». In: *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture Workshops*. 2012, pp. 9–16. DOI: 10.1109/MICROW.2012.12 (cit. on p. 58).

[37] Begul Bilgin, Andrey Bogdanov, Miroslav Knezevic, Florian Mendel, and Qingju Wang. «Fides: Lightweight Authenticated Cipher with Side-Channel Resistance for Constrained Hardware». In: *CHES*. Springer, 2013, pp. 142–158. DOI: 10.1007/978-3-642-40349-1_9. URL: https://www.iacr.org/archive/ches2013/80860199/80860199.pdf (cit. on p. 62).

[38] Cihangir Tezcan. «Analysis of ascon, drygascon, and shamash permutations». In: *International Journal of Information Security Science* 9.3 (2020), pp. 172–187 (cit. on p. 62).

[39] Zhenyu Lu, Sihem Mesnager, Tingting Cui, Yanhong Fan, and Meiqin Wang. *An STP-based model toward designing S-boxes with good cryptographic properties*. Cryptology ePrint Archive, Paper 2023/1023. 2023. DOI: 10.1007/s10623-022-01034-2. URL: https://eprint.iacr.org/2023/1023 (cit. on p. 62).

[40] Léo Weissbart and Stjepan Picek. «Lightweight but not easy: Side-channel analysis of the ascon authenticated cipher on a 32-bit microcontroller». In: *Cryptology ePrint Archive* (2023) (cit. on p. 63).

[41] Claude Carlet. «On highly nonlinear S-boxes and their inability to thwart DPA attacks (completed version)». In: *Cryptology ePrint Archive* (2005) (cit. on p. 67).

[42] Suresh Chari, Charanjit S Jutla, Josyula R Rao, and Pankaj Rohatgi. «Towards sound approaches to counteract power-analysis attacks». In: *Annual International Cryptology Conference*. Springer. 1999, pp. 398–412 (cit. on p. 67).

[43] Emmanuel Prouff. «DPA attacks and S-boxes». In: *International Workshop on Fast Software Encryption*. Springer. 2005, pp. 424–441 (cit. on pp. 67, 69, 70).

[44] Claude Carlet, Annelie Heuser, and Stjepan Picek. «Trade-offs for S-boxes: Cryptographic properties and side-channel resilience». In: *International conference on applied cryptography and network security*. Springer. 2017, pp. 393–414 (cit. on pp. 68, 69).

[45] Mehmet Şahin Açikkapi, Fatih Özkaynak, and Ahmet Bedri Özer. «Side-channel analysis of chaos-based substitution box structures». In: *IEEE Access* 7 (2019), pp. 79030–79043 (cit. on p. 68).

[46] Huizhong Li, Guang Yang, Jingdian Ming, Yongbin Zhou, and Chengbin Jin. «Transparency order versus confusion coefficient: a case study of NIST lightweight cryptography S-Boxes». In: *Cybersecurity* 4.1 (2021), p. 35 (cit. on pp. 69–71).

[47] Yunsi Fei, Qiasi Luo, and A Adam Ding. «A statistical model for DPA with novel algorithmic confusion analysis». In: *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer. 2012, pp. 233–250 (cit. on p. 69).

[48] Stjepan Picek, Kostas Papagiannopoulos, Barış Ege, Lejla Batina, and Domagoj Jakobovic. «Confused by confusion: Systematic evaluation of DPA resistance of various s-boxes». In: *International Conference on Cryptology in India*. Springer. 2014, pp. 374–390 (cit. on p. 70).

[49] Sylvain Guilley, Annelie Heuser, and Olivier Rioul. «A key to success: Success exponents for side-channel distinguishers». In: *International Conference on Cryptology in India*. Springer. 2015, pp. 270–290 (cit. on p. 70).