TÉCNICO LISBOA

Hacking the Systems from Within

André Leitão Nunes da Silva

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisor(s): Doctor Ricardo Jorge Fernandes Chaves
Doctor Aleksandar Ilic

Examination Committee

Chairperson: Doctor Teresa Maria Sá Ferreira Vazão Vasques
Supervisor: Doctor Ricardo Jorge Fernandes Chaves
Member of the Committee: Doctor Alberto Manuel Ramos da Cunha

January 2021

Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

To my grandparents

Acknowledgments

Five years in Técnico, with a one-year break between them, are concluded together with this thesis. It's the end of my cycle towards becoming an engineer; and what a good time it was. Engineering is part of this world both for the good and for the bad, and we need good, creative and ethical engineers to face today's world challenges. I hope this cycle has brought me closer to becoming such one, and I think it did.

There are some people I would like to thank, both for this thesis and for this journey. First of all, I want to thank my supervisors, Ricardo Chaves and Aleksandar Ilíc, for all the attention and help, with the one-hour meetings that quickly became one-and-a-half, with motivation and a smile in everyone's face. It has really been a pleasure working with you.

A big thank you to my mother, for being the engineer of the family, and keeping me inspired and motivated to fight towards all my dreams and goals. Thanks to my father for transmitting me his joy-full and optimistic vibe, and making me see the world in such a way. Thanks to my brothers as well (which, gladly, they will probably never care to read) for being so ambitious and providing such a spirit to our entire family.

I also want to thank my engineer friends, for keeping up for each other and for your rigorous rational and critical thinking, and making the worst tasks and labs a bit funnier; and to my non-engineering friends for reminding me of the most important things of our life.

Finally, a thank you to all the easily-forgotten musicians that bring me joy even when the research results aren't going as expected. Working in silence wouldn't be as fun.

Being thankful is a great feeling. Thank you,

André Nunes da Silva

Resumo

A segurança de informação é um factor fundamental para a nossa sociedade tecnológica. Apesar da maioria dos dispositivos utilizarem algoritmos criptográficos matematicamente seguros, estes têm muitas vezes vulnerabilidades na sua implementação física. Os *ataques de canal lateral* permitem comprometer um dispositivo através de informações provenientes da sua concretização física, como o seu consumo de potência, a radiação emitida ou a duração de uma operação. Por outro lado, devido à importância monetária e ambiental do gasto de energia eléctrica, os processadores actuais incluem sensores de consumo de potência, podendo estes ser lidos pelos utilizadores para controlar ou limitar o consumo energético.

O objectivo deste trabalho é analisar a possibilidade e potencial de um ataque de canal lateral utilizando os contadores de energia do processador para obter as chaves de encriptação utilizadas pelo próprio dispositivo. Este método permite a realização de um ataque sem acesso físico ao dispositivo, ao contrário do uso de um osciloscópio para obter os valores de potência utilizado na literatura científica existente. Em contrapartida, a frequência de amostragem é significativamente inferior e obriga a novas adaptações aos ataques habituais.

Para este trabalho de investigação foi feita uma caracterização das medições de energia obtidas num processador Intel, e foi proposto um ataque de modelação posto em prova contra uma versão simplificada do Advanced Encryption Standard. Finalmente, avaliou-se o seu desempenho tendo em conta diversas características do algoritmo. Os resultados indicam que existe uma fuga de informação detectável e capaz de comprometer uma versão simplificada do algoritmo, mas que utilizando o método proposto é insuficiente para obter as chaves de encriptação do algoritmo real. Este trabalho demonstra também que uma análise futura é necessária para avaliar o potencial desta fuga de informação contra outros métodos e algoritmos.

Palavras-chave: criptografia, ataques de canal lateral, análise de potência, gestão de energia por software, *running-average power limit*

Abstract

Information security is a fundamental aspect of modern technological society. Despite the majority of devices using mathematically secure cryptographic algorithms, it is usual for them to have vulnerabilities in its physical implementation. *Side-Channel Attacks* allow compromising a device using information leaked through its physical properties, such as the power consumption, emanation of radiation or the duration of certain operation. On the other hand, due to the monetary and ecological importance of electric energy spending, modern processors have power consumption sensors included inside, which can be read to monitor or limit the energy consumption.

The goal of this thesis is to analyse the possibility and threat of a side-channel attack using the processors' own energy counters to obtain encryption keys used by that device. This method opens the door to an attack without physical access to the device, in contrary to the usage of an oscilloscope described in current scientific literature. In contrast, the sampling frequency is significantly lower and imposes new challenges to adapt to the current attacking methods.

For this research work, a characterisation of the energy measuring capabilities of an Intel processor was performed and a modulation attack was carried out against a simplified version of the Advanced Encryption Standard. Finally, the performance of the proposed solution was measured under several characteristics of the algorithm. Results show that leakage exists that is detectable and allows compromising the simplified version of the algorithm, but is not enough to obtain the encryption keys of the real version using the proposed method. This work also shows that a future analysis is required to evaluate the threat of this leakage source towards improved methods and algorithms.

Keywords: cryptography, side-channel attacks, power analysis, energy-management software, running-average power limit

Contents

List of Tables

List of Figures

Acronyms

ADC Analog-to-Digital Converter. 18, 37, 39, 63, 64

AES Advanced Encryption Standard. xv, 7, 10, 20, 21, 23, 25, 26, 31, 32, 41, 45, 53, 55, 59–61, 63, 64

CMOS Complementary Metal-Oxide Semiconductor. 15

CPA Correlation Power Analysis. 22, 45

CPU Central Processing Unit. xvi, 2, 10, 11, 16, 17, 24, 29–31, 33, 34, 36, 37, 39, 50–53

DES Data Encryption Standard. xv, 10, 19, 20

DPA Differential Power Analysis. 19, 20, 22, 23, 26

DSA Digital Signature Algorithm. 6

HO-DPA High-Order Differential Power Analysis. 26

MAC Message Authentication Code. 6

MCC Matthews Correlation Coefficient. 46, 50

MSR Model Specific Register. 33–36, 44

PAPI Performance Application Programming Interface. 17, 33–36, 63

PoIs Points of Interest. 23–25

RAPL Running Average Power Limit. 2, 17, 26, 29, 37, 39, 47, 50, 51, 53, 63, 64

RSA Rivest, Shamir and Adelman. 6, 10, 26

SCAs Side-Channel Attacks. 1, 9, 10, 19, 26

SNR Signal-to-Noise Ratio. 18, 30, 31, 39

SPA Simple Power Analysis. 19, 20

Chapter 1

Introduction

There are very two significant aspects among many others that characterise and distinguish our era from the past. These are the amount of available information and the connectivity among people. On one hand, there is an on-going process of digitalisation: information is being stored in digital format, possessing an inestimable price in different aspects besides the economic one. On the other hand, we live permanently connected and in communication with each other through the internet, with efforts being made to bring most devices to a network, forming for example the so-called Internet of Things. All this makes the cyberspace more and more relevant, valuable and real. And like in anything that is valuable and real, threats and risks surge. This way, Information Security rises as an extremely important area of today's society, being crucial for different sectors such as communications, economics, health systems, and even our own entertainment.

Information Security is based upon three fundamental and equally-important pillars: confidentiality, integrity and availability. Confidentiality is the property that imposes that information is only available to authorised users. Integrity of data assures it's accuracy and completeness during it's existence, that is that it can't be modified in an unauthorised way. Finally, availability means that the information is accessible when needed.

These pillars are constantly challenged by the discovery of new vulnerabilities and the creation of new attacks. In order to protect the information from unauthorised parties and therefore ensure confidentiality and integrity, the data is many times encoded or, in other words, encrypted. The area that studies the methods and algorithms to encrypt data is Cryptography. Due to its importance and past study, there are widely-used algorithms that are mathematically very secure. Given the difficulty to break such algorithms, the current strongest attacks don't target the algorithm itself, but it's physical implementation, that is device-dependent and consequently more difficult to protect. These are the so-called Side-Channel Attacks (SCAs). Examples of this kind of attacks use channels like electromagnetic emissions, power consumption or sound to discover the secret information.

Parallel to the increasing technology presence comes the concern of energy usage, both from an environmental point of view and from an economic perspective, especially in big data centers. One example of this aspect is the strategic placement of cryptocurrency mining farms in colder parts of the

globe in order to lower the energy expenses related with cooling the computers. Another, being both an example and a consequence, is the inclusion of power meters in modern processors, in order to monitor and limit their energy consumption. In Intel's processors, this feature is named Running Average Power Limit (RAPL) and is included in all chips since the Sandy Bridge architecture. Bringing this together with the side-channel attacks that exploit the power consumption, the possibility of new threats arise.

1.1 Objectives

The objective for this work is to study the possibility of using the processors' energy measurement counters to create a power analysis side-channel attack. This can be separated into two main goals: first, the information leakage regarding sensitive information of the encryption operations has to be detected through those counters; then, that leakage has to be quantified in order to measure the threat if represents. A parallel goal of this dissertation is the exploration of possibilities for this emerging field.

1.2 Main Contributions

The contributions proposed to tackle those objectives are:

- Characterising the processors' energy counters' measurement capabilities, using the Running Average Power Limit (RAPL) interface of an Intel's Skylake architecture as a practical example;
- Acquisition of power consumption profiles using the mentioned counters;
- Study and evaluation of the required parameters and processing techniques to create a real side-channel attack;
- Design and evaluation of an actual prototype attack;
- Concluding about the threat of energy measurement interfaces towards information security.

1.3 Thesis Outline

This dissertation starts by analysing the background required for side-channel attacks from both a cryptographic point of view and a CPU architecture one. This way, in the first half of Chapter 2 some cryptographic concepts are exposed, namely its different usages, types and algorithms, as well its threats and attacking methods. In the second half, the hardware and functioning of a CPU is analysed with attention given to the origin of the energy leakage explored in this work, and current measurement methods are described.

The state of the art is described in Chapter 3, presenting the origins and important techniques used for side-channel attacks, as well as the countermeasures.

Chapter 4 describes the methodologies of this study and the procedures to follow, including the reasoning behind the chosen paths.

In Chapter 5 the energy measurement interfaces are characterised together with the respective measures in order to define the most promising one. Some properties such as the sampling, update and reading frequencies are defined and clarified, and other parameters are tuned.

Following that analysis, in Chapter 6 a scenario to detect leakage is created and put to test, allowing the construction of methods to process the acquired data and create and differentiate templates. A small comparison is done using the same techniques applied on other systems.

Having successfully detected the leakage, the complexity of the scenario is increased in Chapter 7 in different ways towards a real-case one, and the limits of the defined attack regarding that complexity are drawn. A study is done towards the possibility of bypassing those limits.

Finally, in Chapter 8 a conclusion is presented with the key achievements of this dissertation, together with the future work for this emerging topic.

Chapter 2

Background

2.1 Cryptography

Cryptography is the area of Information Security that deals with protecting sensitive information from unauthorised users. This information can be anything from resting data files, to messages, passwords, among others. This way, cryptography allows the secure transmission of information through insecure channels, in order to ensure confidentiality. Cryptography is also used to digitally sign data or create message authentication codes to provide authentication and integrity. It is used everyday in many forms of technology, like the internet, Bluetooth, mobile telephones, wireless systems, bank ATM's, among others; many times without the average user even noticing it.

The means to implement cryptography are the cryptographic algorithms. These algorithms encrypt data, referred to as plaintexts, by transforming them into incomprehensible ciphertexts. That transformation is based on a secret value, referred to as the key. The inverse operation, called decryption, transforms the ciphertext back into the plaintext, that is the original data. Again, a key is needed to perform this operation. Here, cryptographic algorithms branch into two families: Symmetric Cryptography and Asymmetric Cryptography.

2.1.1 Symmetric Cryptography

In Symmetric Cryptography, the same key is used to encrypt and decrypt. Therefore, it is shared among the parties related to the information. This kind of algorithms are usually more efficient and used to encipher blocks of data. The main disadvantage is that sharing a secret key between different parties can be tricky, especially if there are no secure channels to communicate, which is a realistic assumption.

Encryption algorithms of this family mostly rely on *Feistel* or *Substitution-Permutation* networks to create *block cipher algorithms*, that are therefore combined via different *modes of operation*. The plaintext is divided into fixed-size blocks, and each part is ciphered by the block cipher. The mode of operation provides a chaining rule according to which the blocks encrypted or decrypted depend on the previous or next. This avoids an independent block-by-block operation that would make the ciphertext reveal undesired information about the plaintext.

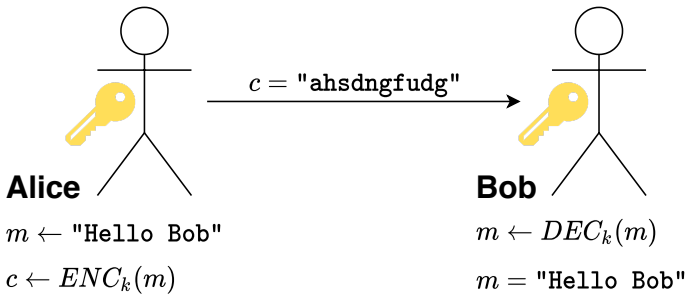

Figure 2.1: Alice sends a message to Bob using symmetric cryptography. The two parties have the same key.

To provide authentication and integrity, a Message Authentication Code (MAC) can be sent together with the message. It consists of a short piece of information generated by the shared key and the message. This way, the receiving party can generate a MAC with the received message and the own shared key, and compare it with the received MAC, detecting any changes to the original message.

2.1.2 Asymmetric Cryptography

Asymmetric Cryptography relies on the use of a pair of related keys, generally referred as the public key and private key. The first key is publicly known, while the second is a secret known only to the corresponding party. This way, if anyone wants to communicate with that person, the messages can be encrypted with the known (public) key, ensuring that only that entity has the private key to decrypt that message. While being less efficient, there is the advantage of allowing communication through insecure channels, since no secrets need to be shared.

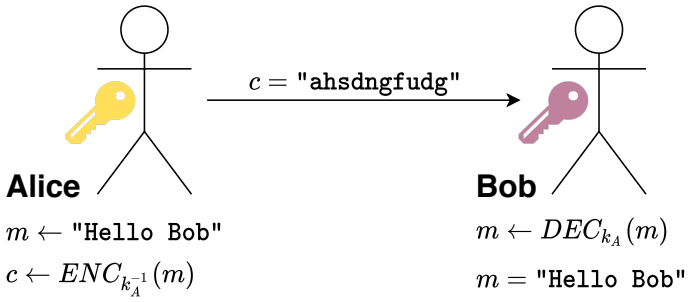

Figure 2.2: Alice sends a message to Bob using asymmetric cryptography. The two parties have two different keys of a key pair.

Asymmetric Encryption algorithms rely on mathematical problems such as the factorization of the product of two large prime numbers (RSA), or the discrete logarithm problem (El Gamal System). DSA is another example of this kind of algorithms.

To ensure authentication and non-repudiation, Digital Signatures can be used. For that, one party can use the private key to encrypt a message. This way, by decrypting it with the corresponding public key, the receiving entity is certain that the message author is correct. Contrary to the Message Authentication Codes, this method also provides non-repudiation since no one else has the private key: the person can

not deny having signed the message.

It is very usual to use asymmetrical algorithms to encrypt a key to send it to the receiving entity, and then use that key as a shared secret to symmetrically encrypt the data.

2.1.3 Advanced Encryption Standard (AES)

The Advanced Encryption Standard (AES) is the most commonly used symmetric algorithm for cryptography. It is a fixed-size block chipher of 128 bits, with different options for the key size: 128, 192 or 256 bits, being therefore often referred as AES-128, AES-192 or AES-256. It consists of a loop of rounds of a *substitution-permutation network*, where the operations are sequentially applied to the original plaintext, called state. This state is viewed as a 4×4 matrix of bytes. The number of rounds depends on the key size, being 10, 12 and 14 accordingly for the 128, 192 and 256-bit versions, with a slightly different last round. For each round there is a *subkey* (or *roundkey*) derived from the main key though a *key scheduler*.

input : 128-bit Block of Plaintext; $128/192/256$-bit Key
output: 128-bit Block of Ciphertext

KeyExpansion;
AddRoundKey;
for 1 **to** $9/11/13$ **do**
    SubBytes;
    ShiftRows;
    MixColumns;
    AddRoundKey;
end
SubBytes;
ShiftRows;
AddRoundKey;

Figure 2.3: Advanced Encryption Standard Overview - Encryption

The performed operations are:

KeyExpansion

RoundKeys are derided from the main key through the AES Key Schedule, which is the algorithm that expands the input key into several. One RoundKey is generated per round, being the first round key equal to the input key.

AddRoundKey

Bitwise Exclusive-OR (XOR) between State and RoundKey: $State \oplus RoundKey$

SubBytes

Substitution of each State byte $a_{i,j}$ based on a *Substitution-Box* (S-Box), in the form $a_{i,j} = S(a_{i,j})$. The S-Boxes are fixed tables, so that this step provides non-linearity to the algorithm.

ShiftRows

Cyclically shifts to the left the bytes of each row n times, being $n = 0, ..., 3$ the number of the row. This step assures that the columns are not encrypted independently.

MixColumns

Combines bytes of each column independently based on a invertible linear transformation in the finite field $GF\{2^8\}$ given by a fixed matrix, in the form

$$\begin{bmatrix} b_{0,j} \\ b_{1,j} \\ b_{2,j} \\ b_{3,j} \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} a_{0,j} \\ a_{1,j} \\ a_{2,j} \\ a_{3,j} \end{bmatrix} \quad j = 0, .., 3 \tag{2.1}$$

This step, together with the previous, provides diffusion to the network.

For decryption, the inverse operations are applied in the reverse order to restore the original plaintext from the ciphertext are represented in algorithm 2.4. The roundkeys order is also reversed, to match the corresponding round.

input : 128-bit Block of Ciphertext; $128/192/256$-bit Key
output: 128-bit Block of Plaintext

KeyExpansion;
AddRoundKey;
InvShiftRows;
InvSubBytes;
for 1 **to** $9/11/13$ **do**
    AddRoundKey;
    InvMixColumns;
    InvShiftRows;
    InvSubBytes;
end
AddRoundKey;

Figure 2.4: Advanced Encryption Standard Overview - Decryption

2.1.4 Cryptanalysis

Contrary to cryptography there is *cryptanalysis*, referring to attacks that try to recover the secret information manipulated by a cryptographic algorithm, which can be either the key, the plaintext or ciphertext, or some key-dependent data that the attacker can use instead of the key. The goal is to improve the time or computational resources it takes to retrieve sensitive information, from the worst case scenario *Brute Force Attack*. This is considered the most trivial cryptographic attack, where every possible key is tried one by one. This attack is unfeasible for most cases, since current algorithms usually use keys with a size of 128 or more bits that make such search an impossible task with even the most recent technology.

Different adversary models can be considered in cryptanalysis, based on the amount of information available to the attacker. For instance, knowing some plaintext/ciphertext pairs, the algorithm or the key size are all variables that are considered. It is generally assumed that the attacker has a perfect knowledge of the algorithm.

Two well known and broad adversary models are the *Black-Box model* and the *Grey-Box model*. Other models exist, but are beyond the scope of this work.

Black-Box Model. The first adversary model illustrated in 2.5 is the more traditional, which assumes that the attacker only has access to the inputs and outputs of the operations, e.g, the plaintexts and ciphertexts. It aids revealing if an algorithm is mathematically secure, independently of the physical implementation. In fact, modern used encryption algorithms are considered or proved secure against attacks based on this model. The already mentioned *Brute Force Attack* is an example of an attack based on this model.

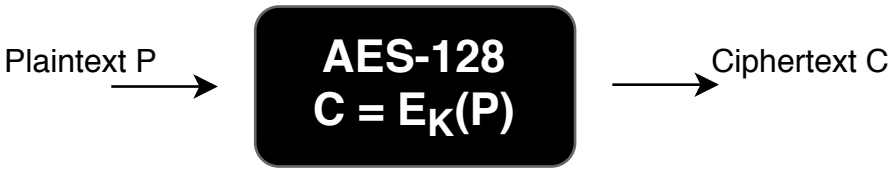

Figure 2.5: Black-Box Model - The adversary can choose plaintexts and/or ciphertexts to try to mathematically break the known algorithm.

Grey-Box Model. In the second model, represented in figure 2.6 the attacker may have access to the physical device where the cryptographic algorithm is implemented. This way, this model illustrates more accurately most real systems, since it is tied to the physical world where devices reside. The attacker may have access to the physical information leaking, such as power consumption, time, or radiation. The resulting attacks are called Side-Channel Attacks (SCAs), introduced in the following section.

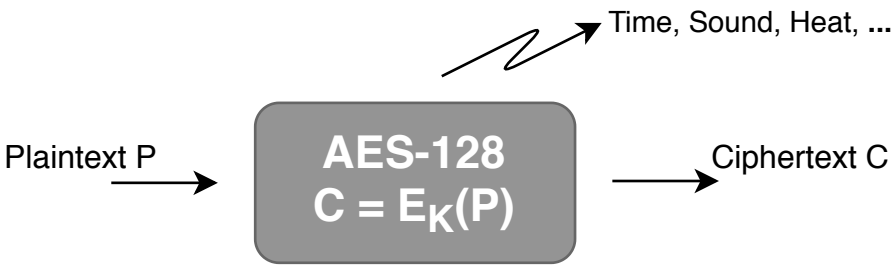

Figure 2.6: Grey-Box Model - In addition to the Black-Box model possibilities, the adversary also has access to the physical implementation of the device.

2.1.5 Side-Channel Attacks

Side-Channel Attacks (or Physical Attacks) are a family of attacks emerging in the late nineties that exploit the weaknesses of the physical implementation of a cryptosystem, instead of attacking the algorithm itself. This uses the fact that while an algorithm might be mathematically secure, it can be leaking

other types of information through its implementation, ignored in the Black-Box model. These attacks can be much more powerful than traditional Black-Box attacks, and present an ever increasing threat to cryptosystems. In fact, most actual cryptographic algorithms have been broken with Side-Channel Attacks. The origin of these attacks date back to 1996, when Kocher [1] broke RSA and Diffie-Hellman systems (asymmetrical encryption algorithms) based on the time it took to perform the operations. Three years later, as described in [2], he used power measurements to break DES (the most commonly used symmetrical encryption algorithm at the time), and proved capable of doing so on other algorithms. This new *Power Analysis* attack has been used across different implementations, such as Smart-Cards [3] , FPGAs [4], to even more recent smartphones [5]. In [6], AES is broken using the timing it takes to encrypt due to cache hits and misses. It is an example of cache-timing attacks. Several other attacks based on caches have been successful [7]. Other interesting type of side-channel attacks abuse the device to make it leak information under subnormal conditions, such as the Fault Injection attack [8]. More attacks exploiting other leakage information exist, like electromagnetic radiation or sound.

Following Mangard, Oswald, and Popp [3], SCAs can be further categorised in different ways, according to the adversary's capabilities, the available equipment and their complexity. They are distinguished between active/passive attacks, and invasive/non-invasive ones, being this categories orthogonal to each other.

Active attacks exploit abnormal behaviours in the devices, sometimes by tempering with it, in order to obtain secret information in a way that would not be possible under normal usage. Examples include overheating the device, variations in the power supply voltage level and irregularities in the clock signal [8]. On the other hand, passive attacks are done under normal behaviour of the device.

Invasive attacks require a permanent action on the device by the attacker. This means that the attacker is able to physically access the chip and its underlying components. This way, direct contact can be established with the chip to recover sensitive information. On the contrary, non-invasive attacks can have access to the external component but cannot modify them in any way. It is restricted to observing the activity, with the possible aid of tools that don't alter the device, such as timers and oscilloscopes.

Given the importance of this family of attacks, under which most cryptographic algorithms have failed to resist when not properly protected, a big concern is given to the corresponding countermeasures. These can range from software changes to big hardware modifications. In this work emphasis is given to power consumption attacks in a non invasive way and the corresponding countermeasures.

2.2 Computational Architecture

The Central Processing Unit is the component of a computer responsible for performing different instructions. This instructions are the steps executed by the processor to perform the operations provided by a computer program in order to do any task. This way, usual programming languages in which programs are written are ultimately translated to a low-level programming language called *assembly*, which has a strong correspondence to what the computational architecture actually does at the architectural level. This assembly code is then *assembled* into executable machine code instructions by an *assembler*, spe-

cific for the processor in question. Those instructions are then executed, performing the desired tasks, such as arithmetic operations or data manipulation. This logic and physical design, together with its organisation and implementation, is called a microarchitecture.

There are significant differences across CPUs, leading to a range of different performances and targeted usages. One important characteristic is the clock rate which refers to the frequency at which the components of the processor operate, in order to synchronise them. This directly reflects on the performance of the processor, as it determines the number of instructions performed per time unit. In an attempt to further increase the processing speed, most computers' have a microprocessor chip with several processors inside them, called *cores*. Figure 2.7 illustrates a diagram of Intel's Skylake chip, containing two cores. This specific client configuration is designed for desktops and personal computers, and there is also a server configuration aimed towards workstations, with up to 28 cores [9]. Alongside the cores which process and perform the instructions, there are other components with different functions. These include the *integrated graphics unit*, which can be seen as a core dedicated to image processing and output, and the *system agent*, which deals with other tasks crucial to the processing such as memory management and control. Surrounded by all this components it's the last-level-cache, which corresponds to a portion of memory shared by all the cores. Finally, the interconnect ring connects the components of the chip.

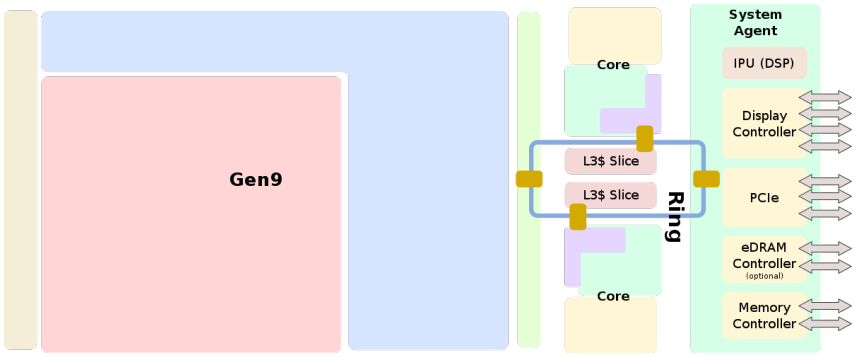

Figure 2.7: Diagram of Intel's Skylake Dual-core Chip (Client Configuration), from *WikiChip.org*. The chip contains two processing cores, the integrated graphics, the system agent and the last-level cache, all connected through a ring connection.

2.2.1 Instruction Set

Intel's processors of the *x86* class support the *x86 instruction set* (corresponding to the *x86 Assembly Language*), which is expanded as new processors and instruction packs are released. Other manufactures have a different instruction set, which serves the same purpose and functions in a similar way. These instructions are represented by a mnemonic that is possibly combined with one or two operands, being then translated to a series of bytes called an *opcode*. The operands correspond to the places where the values used in the operation are stored, called *registers*. The *opcodes* are read and processed by the CPU and generally represent a single executable machine instruction.

In Table 2.1 several examples of assembly instructions are illustrated. The mnemonic specifies the

`MOV EAX, 1`	Moves the value 1 to the register `EAX`
`ADD BX, AX`	Stores in register BX the sum of the values in BX and AX
`NOP`	No Operation
`JMP EAX`	Jump to the value stored in the register `EAX`

Table 2.1: Some examples of simple x86 instructions. Some instructions have no operands, while others have one or two. Note that the storing place of arithmetic operations is over one of the operands.

instruction to be performed, and may require operands or not. To store binary data in order to read and write the result of a command, modern Intel's processors include a collection of 16 64-bits registers. Some of them have a specific usage usually assigned to them, such as the `ESP` that is used to store a pointer of the topmost element in the stack. Others have no such usage, and are free to store and read data.

There are several instruction types that the *x86 instruction set* covers, as illustrated in Table 2.2

Stack	Set up stack to pass parameters to functions, allocate space for data and restore call-return points.	`push, pop`
Data Handling	Data loading and storage, setting a register to a constant value	`mov`
Arithmetic and Logic	Standard arithmetic and logical operations, on integers or floating points, setting status register flags accordingly	`add, div, and, or, shl, shr`
Control Flow	Branch to another location of code, directly, indirectly or conditionally. Call another block of code to implement functions.	`jmp, int, call`
Others: SIMD	Performing an operation on many homogeneous values in parallel, for vector manipulation.	`vmul, psllv`
Others: Cryptographic	Performing encryption/decryption and other cryptographic functions in dedicated hardware	`aesenc, rdrand, sha1rnds4`

Table 2.2: Different types of x86 Instructions and their usage, together with some examples.

2.2.2 Workflow of a Core

In order to perform the different types of instructions in the most efficient way, a series of components exist inside a processor. Figure 2.8 illustrates the block diagram of a core from Intel's Skylake microarquitecture. The core can be separated into two main areas: the *Front End* and the *Execution Engine*, and the auxiliary *Memory Subsystem*.

Front End

The goal of the Front End is to feed the Execution Engine with a stream of operations it gets by decoding instructions coming from memory. The life-cycle of an instruction starts in the *L1 Instruction Cache* after being loaded from memory. At this point, the instructions are organised as *macro-operations (macro-ops)*, which are variable-length instructions. For each clock cycle, a maximum of 16 bytes of code (corresponding to a maximum of 6 *macro-ops*) are fetched from that cache into the *PreDecode*

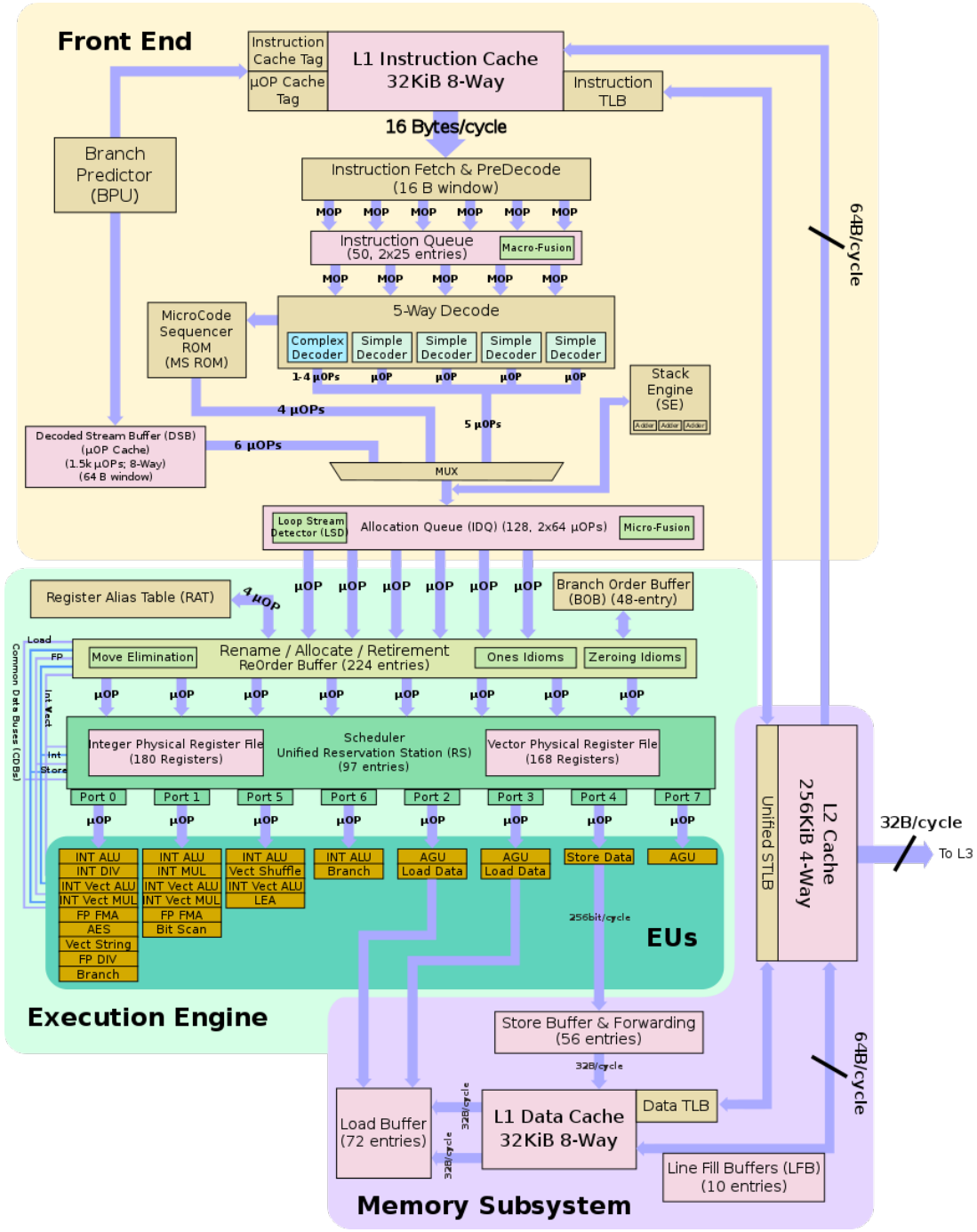

Figure 2.8: Diagram of a Processor's Core, from *WikiChip.org*. The different components are organized into a complex system with the goal of optimizing the security, speed and performance of the instruction flow.

buffer, where the corresponding boundaries are detected and marked. The pre-decoded instructions are consequently delivered to the *Instruction Queue*, which holds a maximum of 50 entries before being sent to the *Instruction Decoder*. In some cases two *macro-ops* are fused into a complex one, saving bandwidth in the remaining pipeline. Five instructions are sent to the decoder per cycle. Its main function is to transform the previous *macro-ops* into regular, fixed-length *micro-ops (µOPs)*, which are sent to the *Allocation Queue*. Some complex *macro-ops* instructions that transform into more than 4 µ*OPs* are

routed through the *Microcode Sequencer ROM* before proceeding, as it proves faster this way than decoding such commands. All this happens in parallel with the *Branch Predictor*, which inspects code further in the byte stream and tries to predict the flow of instructions, and may also send instructions to the *Allocation Queue*. Those instructions are stored in the μ*OP Cache*, and a hit there allows for up to 6 *micro-ops* being sent directly to the *Allocation Queue*, bypassing most of the pipeline and saving important resources and bandwidth. One final component of this part of the processor workflow is the *Stack Engine*, which operates commonly used stack-instructions such as the *POP* or *PUSH* in order to save resources in the back-end.

The *Allocation Queue* acts as an interface between the *front-end* and the *back-end*. Following the previous description, this first works *in order*, meaning that the processing of instructions respect their chronological order in the instruction flow. The same does not apply to the latter, which operates in an out-of-order fashion. This way, the *Allocation Queue* purpose is to effectively dissipate clusters of similar operations that can not be performed simultaneously, ensuring a steady stream of 6 μOPs delivered each cycle to the *Reorder Buffer (ROB)*.

Execution Engine

In the *ROB*, several bookkeeping tasks are done in order to proceed with the instruction. This includes mapping the architectural registers onto the physical ones and allocating resources for data manipulation. The *Register Alias Table* aids in the register renaming by storing where the data is coming from, in terms of previous instructions. At the same time, the *Branch Order Buffer* keeps track of the architectural state in case a roll back is necessary, which is something normal as this processor performs speculative execution. After the *ROB*, instructions proceed to the *Scheduler* where they wait in a queue to be executed. This wait can be due to two reasons: either one required operand has not arrived because it is being loaded or calculated, or the required execution unit is busy. There are eight available ports departing from the *Scheduler*, that lead to different execution units. This means that in the same clock cycle one instruction can be executed from each port, resulting in a total of eight simultaneous instructions assuming they do not require the same execution units. Table 2.3 describes the available operations according to the port of this architecture's scheduler.

After being executed, μOPs are retired in the *Reorder Buffer*, releasing any used resources related to them. This happens *in-order*, so the chronological order of the instructions is re-established.

Memory Subsystem

The memory subsystem controls the load and store requests and ordering. Some buffers exist to support those tasks, namely *Load Buffer*, *Store Buffer* and the auxiliary *Line Fill Buffers*. Different types of memory exist in a processor regarding its size, speed and price characteristics. This allows for a good way of balancing different speed and size requirements while achieving a good performance. Skylake has three levels of data cache, which are denominated L1, L2 and L3 (or Last-Level-Cache) as their size increases and their speed decreases. When a piece of data is requested for processing, the L1 cache is inspected as it is the fastest to deliver the request. If the data is found there, then a *cache hit*

Execution Unit	Port 0	Port 1	Port 5	Port 6	Port 2	Port 3	Port 4	Port 7
Integer Arithmetic/Logic	X	X	X	X				
Vectorial Arithmetic/Logic	X	X	X			X		
Integer Multiplication		X						
Vectorial Multiplication	X	X						
Integer Division	X							
Floating-Point FMA	X	X						
Floating-Point Division	X							
AES Operations	X							
Vectorial String Ops	X							
Vectorial Shuffle Ops			X					
Bit Scan		X						
Load Effective Address			X					
Address Generation					X	X		X
Branch	X			X				
Store Data							X	
Load Data					X	X		

Table 2.3: Operations available at the execution units routed by each port of the Scheduler. While some operations such as Integer Arithmetic can be performed four times during the same clock cycle (using different operands), others such as Data Storage are limited to one.

occurs and the request is processed in the fastest way. However, if the data is not found, a *cache miss* happens and the request proceeds to the L2. Here the same procedure happens, and so on. When the request is found, it is stored in a closer cache to allow future access, under certain restrictions and control. Data caches allow for much faster processing because it is common for the same data to be requested several times.

2.2.3 Energy Consumption

Digital circuits operate based on two levels of discrete voltage levels, labelled *0* and *1*. To implement the Boolean (binary) logic, operations are done through *logic gates*. These are electronic devices that implement some Boolean function, producing an output (0 or 1) based on its input. They serve as the building block for more complex devices, such as multiplexers, registers and memory, until complete microprocessors. Despite existing different ways of implementing those logic gates, the Complementary Metal-Oxide Semiconductor (CMOS) transistors technology is by far the most used. In fact, in 2011, 99% of all integrated circuit chips were fabricated with this technology [10].

The relevant advantage of CMOS over the concurrent technologies regarding this work is the low power dissipation. This is possible due to the complementary pull-up and pull-down network organisation, which is arranged such that when one is activated, the other is disabled. This results in a static consumption power near zero, as the path between the power source and the ground is not connected during idle moments. The main source of power consumption is then the dynamic part, which corresponds to the energy that is consumed during the transitions of state. These transitions result in brief moments when both networks conduct and the connection between the power source and the ground

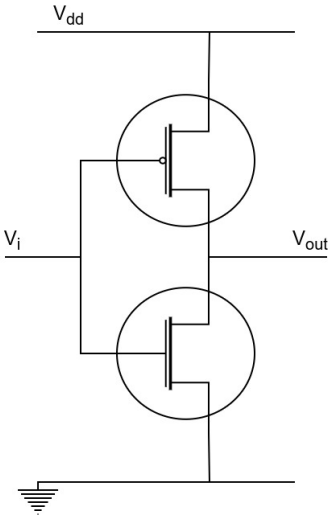

Figure 2.9: CMOS Inverter Gate. When the top transistor is activated the bottom one is disabled, disconnecting the path between the power source and the ground.

is established, resulting in energy dissipation. Other minor sources of power consumption include the transistor leakage currents, that correspond to the energy that flows through a boundary seen as insulating, and short-circuit power consumption, that is related to brief short-circuits across transistors during a state change of a logic gate.

Power consumption sources are complex and depend on many factors, from the physical properties of the transistors to external conditions like the temperature, being therefore difficult to model precisely. It is known that the dynamic and short-circuit fractions of the power consumption are proportional to the clock frequency, while the leakage currents are proportional to the supply voltage. Despite that, the dynamic power consumption, being only consumed during transitions of state of a logic gate (and not on every clock cycle) is dependent on the data being manipulated. This includes the flow of instructions being processed in the whole processor, and the operands used in the execution units, stored in registers. Consequently, if the same algorithm is executed on different pieces of data, the power consumption is different due to the different values of the registers. For the same reasons, two different algorithms being executed with the same data will also have a different power consumption associated. .

This correlation between the manipulated data and the power consumption provides a chance for the exploitation used in the already mentioned side-channel attacks. It should also be noted that the actual value of power consumption is usually irrelevant, as it is the relative differences that leak the sensitive information.

2.2.4 Software-based Energy Measuring

With the growing amount of data centers and devices, together with the importance of saving energy or controlling its expenses, power consumption becomes a critical metric in the design and usage of electronic systems. With high-performance computing machines having hundreds or thousands of cores, reducing a few Watts per CPU quickly adds up to significant power, cooling and monetary savings [11]. This has led to the introduction of new components into CPUs that allow energy and performance

measurement, in order to measure or limit the consumption as necessary. Despite their importance, their consequences on information security are not completely understood[12].

In Intel's CPUs, this feature is covered by Intel's Running Average Power Limit (RAPL), which consists of a set of measuring sensors and counters, as described in the Intel Software Developer's Manual [13]. It is included in their CPUs since the *SandyBridge* microarchitecture, with slight differences across the many subsequent ones.

RAPL stores the energy values in different registers, according to the domain of the measurement. The domains refer to different parts of the processor:

- **Package** - Measures energy consumption of the entire chip. This includes all the cores, integrated graphics, and *uncore* components such as last-level-cache operations and memory controllers.
- **Power Plane 0 (PP0)** - Measures energy consumption of all the cores.
- **Power Plane 1 (PP1)** - Measures energy consumption of the integrated graphics.
- **DRAM** - Measures energy consumption of the random access memory attached to the integrated memory controller.
- **Psys** - Measures energy consumption of the Package Domain, plus PCH, eDRAM, among others.

The units for the power, time and energy readings vary according with the microarchitecture, and are found in the register *MSR_RAPL_POWER_UNIT*. For example, in order to convert an energy measure to Joules, it has to be multiplied by $1/2^{ESU}$, with ESU being the value represented in the bits 8 to 12 of that register. For the Skylake microarchitecture studied in this work, the ESU has the value 14. This way, the measures are transformed to Joules after being multiplied with $1/2^{14} = 61$ μJ. This corresponds to the minimum difference that two power/energy samples can have, and is refereed as the resolution. Other architectures such as the Haswell have a resolution of 15.3 μJ. Notice that a smaller resolution allows for more accurate measures, as smaller differences in the input signal can be detected. This way, the raw integers stored in the register have to be multiplied by the resolution to be transformed in Joules.

In order to read those registers, an interface has to be used. The most common one is the Intel's Performance Application Programming Interface (PAPI) [14]. There are other options, as well as the possibility of customising an interface to read the driver. The comparison of different interfaces is presented in Chapter 5.

When dealing with the sampling of continuous signals such as the power consumption, it is important to consider the analog-to-digital conversion limitations and properties. Unlike the physical world, digital systems always deal with discrete values, as a result of their own representation in a stored number. This way, digital signals, and therefore any sampled signal, have a discrete time and a discrete amplitude. Such transformation from a continuous environment to a discrete one involves the *quantization* of the input signal, that is approximating real-world values with a digital representation of them, introducing necessarily an amount of *quantization* error, as represented in figure 2.10.

Another consequence of a discrete time environment is that the real signal is sampled periodically, limiting the allowable bandwidth of an input signal. The frequency at which the signal is sampled is called

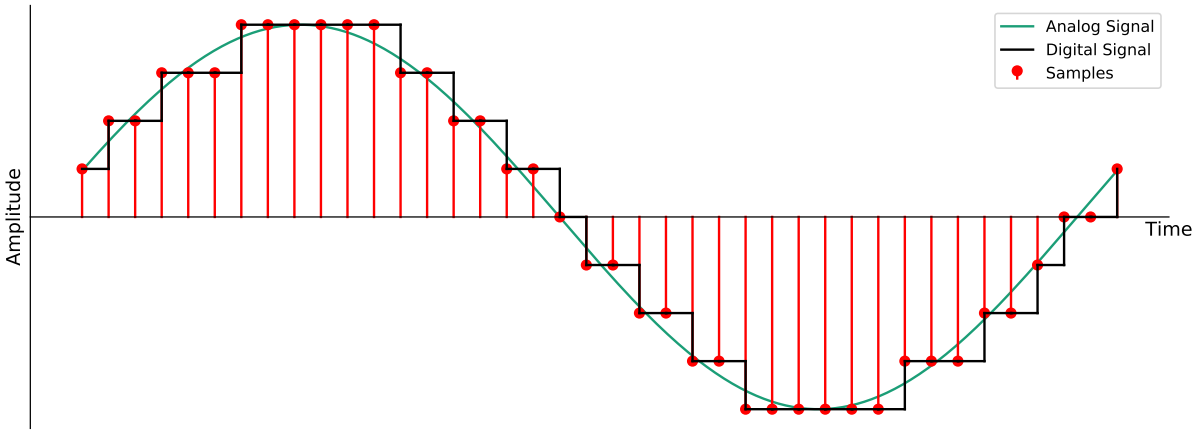

Figure 2.10: Quantization of a Signal. The digital signal differs from the analog input as it uses only certain values - the quantization levels.

the *sampling rate*. This constitutes two main characteristics of an Analog-to-Digital Converter (ADC). In order to allow for a correct reconstruction of a sampled signal and assure that no information is lost, the Nyquist Theorem states that the sampling rate must be at least two times the maximum frequency of a signal. The phenomena of a reconstructing signal differing from the original one is called *aliasing*.

ADCs have some important characteristics that indicate their overall behaviour, such as the already mentioned sampling rate and resolution, and the Signal-to-Noise Ratio (SNR), that corresponds to the ratio between the power of the input signal and the power of the background noise. In the usual cases where the noise cannot be separated from the signal, the SNR is given by

$$SNR_{dB} = 10 \log_{10} \left(\frac{P_{S+N} - P_N}{P_N} \right) \tag{2.2}$$

where P_{S+N} is the power of the meaningful signal plus noise, and P_N is the power of the noise alone. The power of a digital signal can be calculated with

$$P_s = \frac{1}{N} \sum_{k=0}^{N-1} |s(k)|^2 \tag{2.3}$$

with N being the number of samples of the signal s.

This SNR is also influenced by a group of other characteristics like the *non-linearity* or the *jitter*.

The *non-linearity* affects all ADCs and refers to the errors caused by the physical imperfections that result in the output signal deviating from a linear function. The *jitter* comes from the small time differences of each clock cycle that lead to some uncertainty in the sampling interval. While non-existing at DC measures, its effect is significant at high-frequency acquisitions.

Chapter 3

State of the Art

3.1 Power Analysis

Power Analysis is the branch of Side-Channel Attacks in which the channel used for exploitation is the power consumption of a device. Given that the instantaneous power consumption is usually dependent on the data being processed and on the instructions being performed, a proper analysis of the power consumption during the encryption or decryption operations can reveal leaked sensitive data, including the secret key. The first Power Analysis attack was published in 1999 by Kocher, Jaffe, and Jun [2], where the power consumption was measured with an oscilloscope and used to successfully break the DES algorithm, that was the most widely used encryption algorithm at that time.

Most Power Analysis attacks are based on acquiring *power traces*, which are a set of consumption measurements taken during the execution of one cryptographic operation. This results in a sampled power consumption plot, as the one in Figure 3.1.

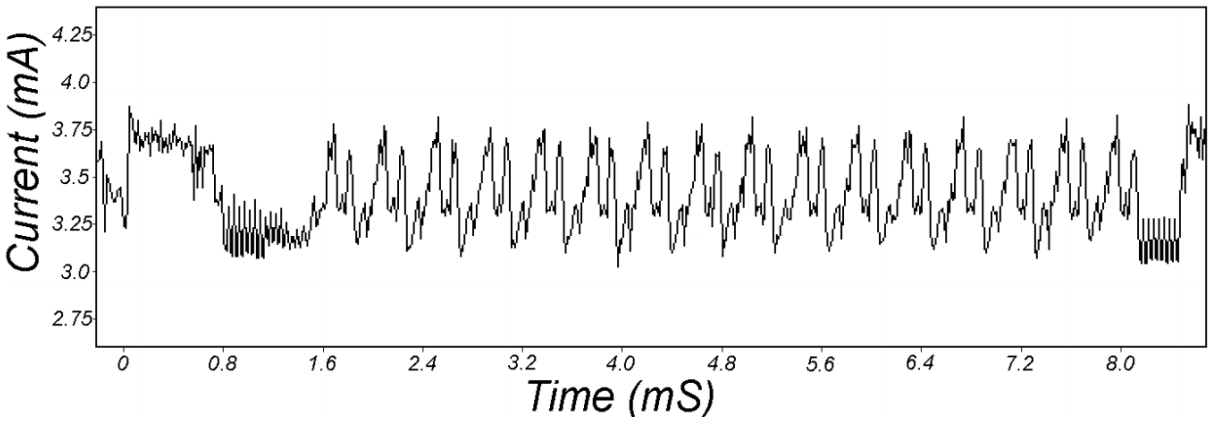

Figure 3.1: Power trace during one encryption with the DES algorithm on a Smart-Card, retrieved from [2]. The pattern of 16 similar oscillations represent the 16 rounds of the DES algorithm.

Different techniques exist to extract sensitive information from the power traces, branching Power Analysis into Simple Power Analysis (SPA) and Differential Power Analysis (DPA), or other types of attacks, such as the Template Attacks.

3.1.1 Simple Power Analysis

The simplest kind of Power Analysis Attacks are grouped under the name Simple Power Analysis. They exploit the information available at one or few power traces by visually interpreting it. For instance, in Figure 3.1, there is a clear pattern repeated 16 times, that correspond to the 16 rounds of the DES. That alone already provides a clue on the algorithm used.

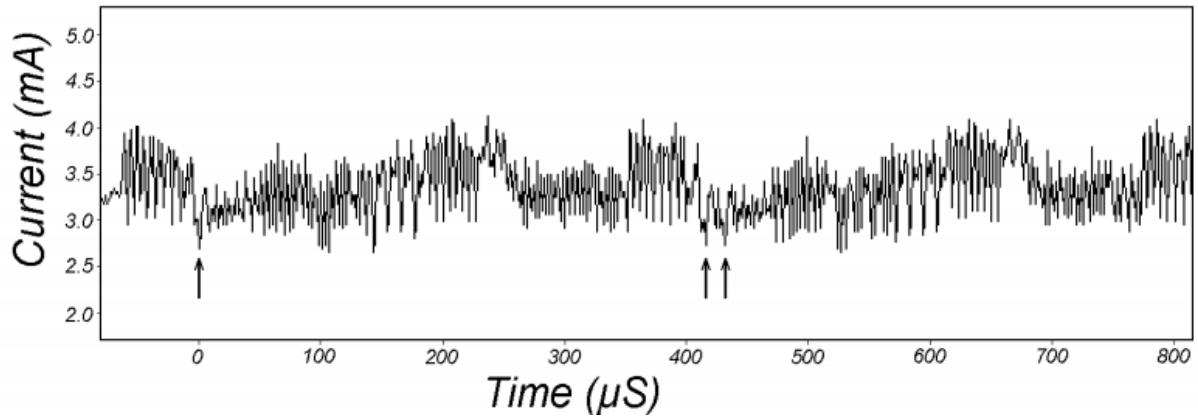

Figure 3.2: Second and third rounds of DES, as a detail of 3.1 retrieved from [2]

In Figure 3.2 a more detailed view is shown, focusing on the second and third rounds of DES encryption. More details are visible, as we can see pointed by arrows: a certain operation (in this case a register rotation) was performed once in round two and twice in round three. Also, given the high power consumption of the conditional jump operation, by studying the flow of the algorithm and comparing with the analyzed plot one can guess the respective condition bit value. Other recognizable operations exist that leak data exploited through this technique.

Being SPA a not-so-recent technique, modern technology can easily evade this threat. Firstly, a smaller power consumption in components together with higher clock rates greatly increase the difficulty of interpreting the plots visually; and secondly, avoiding using sensitive information as a branching condition will obstruct most of the information.

3.1.2 Differential Power Analysis

A much more powerful form of attacks published in [2] consists of a statistical approach, usually referred to as Differential Power Analysis (DPA). In this case, the idea is to use information collected on many acquired consumption traces. It is used against algorithms in which the secret key K (or a subkey that derives from the secret key) is split in small parts, allowing a divide and conquer approach. Such is the case of most block cipher algorithms such as DES and the widely used AES.

To perform the attack, N power traces $T = T_1, ..., T_N$ are acquired with W samples each, corresponding to the execution of the algorithm with N different plaintexts $P = P_1, ..., P_N$, and the same unknown key K. Let $T[j]$ be the j^{th} sample of a given trace T. With the knowledge of the algorithm, the attacker can identify a intermediate bit that depends both on a small number b of key bits, K^*, and on

the known plaintext P (or ciphertext). The number b must be small enough to allow a practically feasible brute force search with size 2^b. Modelling the computation of that bit during the algorithm, a function is designed, taking as input those bits of the key and the respective plaintext. This function is called the *selection function*, with the form $D(P, K^*)$.

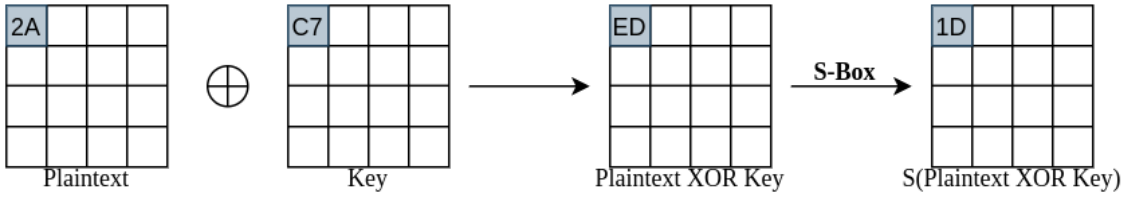

Figure 3.3: Selection function designed for AES to compute the first byte of the state after the *SubBytes* operation in the first round. The figure illustrates how it depends only on the first byte of the Plaintext, $P[1]$, and on the first byte of the Key, K^*. For this algorithm, $b = 8$ is a good choice since the state is treated byte by byte. Note that the S-Boxes are fixed tables, and that the roundkey of the first round of AES is the secret key itself. The resulting function is $D(P, K^*) = S(P[1] \oplus K^*)$, with S being the S-Box substitution.

The attacker then separates each power trace T into two sets S_0 and S_1 according to the value of the most significant bit (MSB) of the *selection function* output, with the respective plaintext P and a given key part K^* as input. Finally, a *distinguisher* Δ_D is applied to them. The one used originally by Kocher et al. consisted on calculating the trace corresponding to the difference of the averages of each set S_0 and S_1 as:

$$\Delta_D = \bar{S_0} - \bar{S_1} \tag{3.1}$$

This can be done sample by sample, assuming that the traces are synchronized in time, as

$$\Delta_D[j] = \frac{\sum_{n=1}^{N} MSB(D(P_n, K^*))T_n[j]}{\sum_{n=1}^{N} MSB(D(P_n, K^*))} - \frac{\sum_{n=1}^{N}(1 - MSB(D(P_n, K^*)))T_n[j]}{\sum_{n=1}^{N}(1 - MSB(D(P_n, K^*)))} \tag{3.2}$$

If the guessed key part hence designated as K^* is wrong, then the output Δ_D will be a small noise near zero, since the result of the selection function D was different from the target for about half of the plaintexts. There is no correlation between the selection function and the actual computation done by the target device. Actually, as we increment the number of traces N, the better the averages will cancel each other and the closer Δ_D will be to zero.

However, if K^* is correct, the output of the selection function will be the same as the one computed by the device for all the plaintexts, creating a correlation. As a result, Δ_D will approach the effect of the target bit on the power consumption as N increases. The plot will be flat with spikes in the zones where D is correlated to the processed values. The attacker can then identify the correct key-part K^* by trying the method with all the 2^b possibilities, and evaluating the computed Δ_D.

By knowing one part of the key, the remaining bits can be guessed to allow discovering the respective missing parts. For this, it is used the 2nd byte of the key, and so on. The separation of the keys into different parts is what reduces the complexity of these kind of attacks to a feasible value. Given the case of AES-128, a brute force search for the whole 128-bit key would require 2^{128} tries. However, splitting it

into 16 parts (1 byte each) and searching for each one individually on a space sized $2^8 = 256$, requires $16 * 256 = 4096$ iterations.

After this research work, many other works were proposed to complete and diversify this technique. Several other distinguishers have been compared [15, 16], such as the *Mutual Information Analysis* [17], the *Kolmogorov-Smirnov* [18] and the *T-Test* [19]. One of the them, probably the most widely used, is the *Pearson Correlation Coefficient*, explained in the following section.

3.1.3 Correlative Power Analysis

Correlation Power Analysis (CPA) uses the Pearson Coefficient of Correlation instead of the Difference of Averages as the *distinguisher* applied to the power traces set. It has numerous advantages, including the need for a smaller number of power traces [20].

Imagining the same situation as above, where there are N power traces $T = T_1, ..., T_N$ with W samples each. Let $T_n[j]$ be the j^{th} sample of the n^{th} power trance, with $1 \leq n \leq N$ and $1 \leq j \leq W$. Corresponding to each power trace are N plaintexts $P = P_1, ..., P_N$, so that the trace T_n was measured while operating the P_n plaintext. Consider as well the same selection function $D(P, K^*)$, that computes an intermediate value with the plaintext, based on a key part K^* of b bits.

For CPA, a *Power Model* is defined for estimating the power. Because the dynamic power consumption depends on the number of bit changes, the *Hamming Distance* can be used for this model: the Hamming Distance (HD) between two binary numbers is the number of different bits, position-wise. This way, the *Hypothetical Power Consumption* can be computed with $H_{n,k} = HD(D(P_n, K_k^*))$ for the n^{th} trace/plaintext pair and k^{th} possible key part, with $0 \leq k < 2^b$.

Then, for each sample j and considered key part K^* the Pearson Correlation Coefficient $\rho_{j,k}$ is calculated to measure the correlation between the *Hypothetical Power Consumption* and the real measures:

$$\rho_{j,k} = \frac{N \sum_{n=1}^{N} T_n[j] H_{n,k} - \sum_{n=1}^{N} T_n[j] \sum_{n=1}^{N} H_{n,k}}{\sqrt{N \sum_{n=1}^{N} T_n[j]^2 - (\sum_{n=1}^{N} T_n[j])^2} \sqrt{N \sum_{n=1}^{N} H_{n,k}^2 - (\sum_{n=1}^{N} H_{n,k})^2}} \tag{3.3}$$

The Pearson Correlation Coefficient outputs a value between -1 and 1. Higher absolute values mean that the two data sets compared have the best correlation, while a value of zero means that there is no correlation at all. The sign is not relevant for this case. Since the used data sets are the *Hypothetical Power Consumption* and the real power traces, the best correlation will most likely represent the correct key part. There is no need to separate the coefficients by sample, because time information about the moments in which that correlation manifests is not relevant.

Similar to the DPA, after matching one part of the key, the same method can be used to discover the remaining key parts.

3.1.4 Template Attacks

A much different family of Power Analysis attacks are the Template Attacks, introduced by Chari, Rao, and Rohatgi [21] in 2003. These attacks are very strong in an information theoretic sense, because

all the information leaking is potentially used. They usually require much less samples than correlation attacks. In order to create the templates for the attack, the attacker needs to have an exact copy of the targeted device to create a model of the its noise leakage. This is effortlessly achieved when the device in question is mass-produced and easily accessible. A typical template attack is then composed of two phases:

1. **Profiling Phase (or Training Phase)**. The attacker uses his replica of the targeted device to model the leakage regarding different operations.
2. **Attack Phase**. The power traces acquired for the targeted device are processed and compared with the model constructed in the first phase to obtain the secret key.

Profiling Phase

It is customary in signal processing to model a sample as a combination of a value and noise. This noise is best modelled as a random sample drawn from a noise probability distribution. In this phase, the goal is to develop a Multivariate Gaussian Model for the noise (power consumption) associated with different operations as accurately as possible.

To start, a large number of power traces are gathered using the clone system which is under full control of the attacking party. This way, this number is limited only by the time and available storage. The traces correspond to the power consumption of the device while performing the operation being modeled. While this can be used to distinguish any type of operation, the idea in template attacks is to consider each operation as performing an encryption with some different bits, depending on the key and plaintext. For example, reminding the selection function used also in DPA, one can consider the output of the AES S-Boxes to differentiate between operations. This way, it can be considered $K = 256$ different operations, where operation O_j is defined as encrypting a plaintext byte Pt_x and respective key part k^* such that $S(Pt_x \oplus k^*) = j$, with S being the S-Box substitution. The set of power traces acquired while performing O_j is represented by S_j. It is common practise to choose the plaintexts and keys randomly and mapping the corresponding trace to the correct set.

Modeling an entire power trace t_i with W samples, $t_i[1], ..., t_i[W]$, requires a W-dimension distribution, which proves unfeasible with a realistic value for Wa, in the order of thousands. This way, and because most points of the power trace are not relevant since they are not directly affected by the key, a group of N Points of Interest (PoIs) representing the critical and key-dependent moments of the trace are selected. Various techniques are used to select pick those points, as well as other methods to compress the data, as discussed later.

Having chosen N PoIs, $P_1, ..., P_N$, the power traces can be transformed by keeping only the positions related to those points.

$$t = t[1], ...t[W] \in \mathbb{R}^W \quad \mapsto \quad \dot{t} = t[P_1], ...t[P_N] \in \mathbb{R}^N \tag{3.4}$$

Finally, the template parameters can be computed. These are the average, $\mu_k \in \mathbb{R}^N$ and the covariance matrix $\Sigma_k \in \mathbb{R}^{N \times N}$ defined as

$$\mu_k = \frac{1}{|S_k|} \sum_{\dot{t} \in S_k} \dot{t}, \quad \Sigma_k = \frac{1}{|S_k| - 1} \sum_{\dot{t} \in S_k} (\dot{t} - \mu_k)(\dot{t} - \mu_k)' \tag{3.5}$$

The template of the operation k is defined by the tuple (μ_k, Σ_k).

Points of Interest Selection and Data-Compressing Techniques

Different techniques exist in order to reduce the templates to a reasonable size. In fact, using the information of all the samples would waste a lot of computing time and power since most of them don't have any relevant information to the attack. It is then necessary to select that relevant PoIs. Identifying those points leads not only to a serious reducing in data size, but also leads to discovering which operations of the algorithm under attack leak the most.

The original method, introduced in Chari et al. [21], consists of computing the sum of the pairwise differences between the average signal of each template, Δ, and select only the N points at which large differences show up. In [22] an improvement is made by squaring the sums. This avoids hiding important peaks due to different signals, resulting in

$$\Delta = \sum_{u=1,v=1}^{K} (\mu_u - \mu_v)^2, \quad u \geq v. \tag{3.6}$$

with the resulting peaks of Δ identifying the points at which the biggest differences show up. Since the sampling rate is usually higher than the clock frequency, it is normal that Δ shows clusters of high points corresponding to the same operation of the cipher. This way, a usual method to contour this limitation is imposing a certain spacing between each chosen PoI.

In a more advanced method is also proposed, involving the T-Test to successfully detect the PoIs with noisy signals [22]. The T-Test is a statistical hypothesis test that distinguishes two data sets (i, j), which essentially compares the distance of the corresponding means (m_i, m_j) and their variability (σ_i^2, σ_j^2). Among various implementations, depending on the similarity of the variances and sample sizes (n_i, n_j) of the data sets, the Welch's T-Test is given by

$$t_{value} = \frac{m_i - m_j}{\sqrt{\frac{\sigma_i^2}{n_i} + \frac{\sigma_j^2}{n_j}}} \tag{3.7}$$

and is used when the different sample sizes and variances are a possibility. This way, the PoIs can be detected with

$$\Delta = \sum_{u=1,v=1}^{K} \left(\frac{m_u - m_v}{\sqrt{\frac{\sigma_u^2}{n_u} + \frac{\sigma_v^2}{n_v}}} \right)^2, \quad u \geq v \tag{3.8}$$

In order to reduce the data size in another way, Elaabid et al. [23] exploited the fact that the power dissipation in a CPU can be proportionally approximated by the Hamming weight of the computed data, in order to reduce the number of templates required. Therefore, it is possible to model a sensitive part of the algorithm and create a template for each possible Hamming Weight of a state of that model. In

AES, the diffusion of the S-Box makes it a good part to model. With a key part of 8 bits, this technique uses 9 templates instead of $2^8 = 256$. Despite this only allowing the discovery of the Hamming weight of the key part, its exact value can be recovered by repeating the attack with other plaintexts.

Attack Phase

Having computed the templates (μ_k, Σ_k), it is necessary to establish how to discover the subkey $\hat{k}$ which resulted in the set of traces, $\hat{S}$, acquired in the targeted machine. Recalling that the leaking noise is approximately modeled by a multivariate normal distribution, the probability distribution of the noise occurring from operation O_k is given by the N-dimensional normal distribution $p_k(.)$ where the probability of observing a noise vector z is

$$p_k(z) = \frac{1}{(2\pi)^N |\Sigma_k|} exp\left(-\frac{1}{2} z' \Sigma_k^{-1} z\right), \quad z \in \mathbb{R}^N \tag{3.9}$$

with $|\Sigma_k|$ representing the determinant of Σ_k, and Σ_k^{-1} its inverse.

This way, to classify a power trace $\hat{t}$ from the set $\hat{S}$ a maximum likelihood hypothesis test is performed. For each $k \in [0, K[$, the noise in $\hat{t}$ is extracted at the N PoIs, yielding a noise vector $n_k(\hat{t})$ with

$$n_k(\hat{t}) = t[P_1] - \mu_k[P_1], ..., t[P_N] - \mu_k[P_1] \in \mathbb{R}^N. \tag{3.10}$$

Then, the probability to observe such a noise vector can be computed using Equation (3.9). Consequentially, the hypothesis k that maximises that probability is the best candidate for the observed trace $\hat{t}$. When more than one trace is available, the probability to be maxed is found with,

$$P_k = \prod_{\hat{t} \in \hat{S}} p_k(n_k(\hat{t})). \tag{3.11}$$

Other ways to correctly identify the template exist. For instance, [24] uses Support Vector Machines, a family of machine learning algorithms, to tackle this challenge. After acquiring one byte of the key, the power traces can be mapped to new sets according to the next key and plaintext bytes, and the process is repeated.

3.1.5 Countermeasures and Counter-countermeasures

Given the importance of these kind of attacks, a great effort has been made in order to find countermeasures to prevent them [25]. Different categorisations can be done on this countermeasures, that ultimately divide them into Software-based / High-level versus Hardware-based / Low-level [20]. In [26] a Medium-level, as something between the two extremes, is also presented.

Hardware-based / High-Level countermeasures relate to the root of the problem: the physical leakage of the device. They focus on securing that no information leaks, and therefore no more measures are necessary at a software-level. For this reason, they are also known as *hiding* [27]. For instance, in Popp and Mangard [28] is explored the possibility of a *dual-rail* technique, which is a low-level countermeasure that consists of doubling the wires transporting the information bits in order to spread, in addition to the

original signal, the corresponding complementary. This way, the power consumption is in theory constant and independent from the manipulated data. Another example is adding a noise source component, so that there is no noticeable correlation between the power consumption and the manipulated data. This type of countermeasures have the advantage of being completely independent from the executed algorithms at the cost of being directly dependent on the inherent technology, and hard to quantify [26]. They can be very effective in reducing the leakage, but usually lead to higher prices or reduced performance.

On the other hand, there is the Software-based / Low-level approach, that consists of algorithmic countermeasures which are less dependent on the device. For example, *masking*, being probably the most deployed side-channel countermeasure [25], consists of splitting the intermediate values of cryptographic computation into randomised shares to avoid dependencies between these values and the power consumption. Different methods exist for the randomisation, and the designer can also choose the number of shares, known as order, being 2^{nd} order the most usual choices. This usually mitigates the risk of simple DPA attacks, but more complex High-Order Differential Power Analysis (HO-DPA) with a high enough attack order can still present a threat [29]. These are attacks that analyse together different statistical properties of the measures. For example, multiple selection functions can be modelled in order to exploit information of multiple points in time. This allows to successfully breaking cryptosystems with noisier traces or bypassing the already mentioned *masking*.

Despite the proposed countermeasures to SCAs, there are two main reasons for it to still present a significant threat. In the first place, like in most areas of Information Security, new attacks are discovered and deployed that totally or partially bypass the current defences. It is a race that never ends between new attacks and new defences, like seen with the use of HO-DPA to bypass *masking*. The other reason is that countermeasures, even software-ones, usually cause an worst performance in terms of computational time and memory, require higher hardware space and cost and in general bring more complexity.

3.2 Attacks based on energy measurement acquired by software

Being a relatively new area of interest, there is not a big collection of articles regarding vulnerabilities introduced by Intel's RAPL and similar features. However, the existing research already confirms that some threats exist [12, 30]. Mantel, Shickel, Weber, and Weber [12] successfully distinguish two secret keys of the Rivest, Shamir and Adelman (an asymmetrical encryption algorithm) using only RAPL measures through a modified template attack. They specifically target the Bouncy Castle library, and show that 25 encryptions already give the attacker a 99% change of success with a Bernoulli-trials approach.

Paiva, Navaridas, and Terada [30] use RAPL in an area other than cryptography, by using the DRAM power consumption to create covert channels, which are communication channels that allow a process to communicate with another one in a supposedly isolated environment. This demonstrates another level of relevant threats created by software-based energy measurement features.

Finally, other relevant work was done by O'Flynn and Dewar [31] by breaking AES using an on-board

ADC on a SAML11 microcontroller without any external measuring equipment. White this work uses an on-device oscilloscope present in that board and not power measuring counters like the ones of desktop computers, it consists of a type of Power Analysis attack without physical access to the device.

Chapter 4

Proposed Method

This work studies the possibility of exploiting the correlation between the power consumption and the processed data in a CPU using the energy measuring capabilities provided by the energy counters, in particular the cryptographic operations. Note that in usual power attacks an oscilloscope is used to gather the so-called power traces, its important to note that such attacks are not performed against usual computers, but target simpler microprocessors, such as credit cards, FPGAs or smaller devices. There are two main reasons for that. First, it would be very difficult, if at all possible, to obtain power traces of such a processor with an oscilloscope, since an entire complex system is in that same chip. Secondly, in terms of a real case it is difficult to have physical access to a targeted computer and an oscilloscope. Instead, in this scenario the energy counters from the processor are used to provide the power consumption samples. This has the advantages of not requiring physical access to the processor, as these counters are read via software. Another advantage is that by selecting the respective power plane, the readings are from the cores only, excluding the other components of the chip. On the other hand, these counters provide a worst sample resolution and a sampling rate significantly lower than the clock frequency.

An overview of the conceptual architecture of a proposed attack is illustrated in Figure 4.1, influenced on the existing template attacks. This approach can be separated into the three following different stages, studied in this thesis.

Interface - characterisation and measuring capabilities of the interface that connects the user to the energy counters in the CPU and its respective readings. In the specific example of Intel processors, this corresponds to RAPL.

Processing - Processing of the measured data and methods to create templates or attacking traces that model the power profile of a certain operation to certain data.

Power Analysis - Proposed methods and algorithms to match or differentiate the templates and an attacking trace in order to obtain information about the corresponding operation, and attacks that exploit that method to obtain sensitive information.

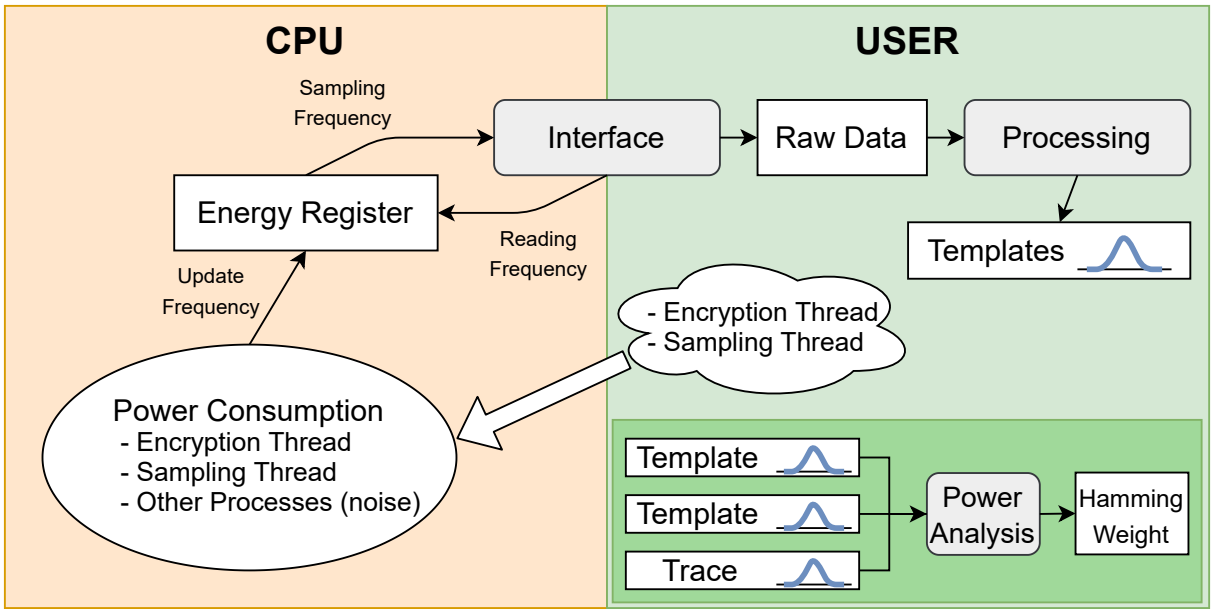

Figure 4.1: Conceptual architecture of a possible attack. An user triggers the encryption and sampling threads, and detects its power consumption through an interface that reads the energy register. That information is processed to create templates, which are used to match a trace obtained in a similar way on the machine under attack, in order to discover information about the key.

4.1 Interface Characterization of Measures

Prior to acquiring measures with any measuring tool it is crucial to know some of its specifications, namely the resolution, the sampling rate and the SNR. For our case of energy counters inside a CPU, one more property becomes relevant, that is the noise produced by the operation of reading the registers and providing its value to the user. This is a result of both the actions - reading the registers and encrypting the data - being performed under the same power plane, that is the cores of the processor. To quantify this noise, a *reading frequency* can be measured, that corresponds to the maximum amount of register reads that the interface can do in a time interval. This way, it is expected that an interface that uses a higher amount of computational resources to read the registers storing the energy values has a lower reading frequency. The importance of this parameter is increased by the fact of it being necessary to actively read the register in order to know when it updates. Also, because the experiments are to be made in a single thread in order to avoid context switching in the cores, the time spent reading the register means a period of time in which encryptions are not taking place.

It is therefore of maximum importance to minimize the computational resources used while accessing the register, and attention should be given to some details of the implementation. For example, the read values should be stored in a static array in order to avoid allocating memory during run time, and using computationally expensive functions like writing to a file should be done at the end of the acquisition. Also, another aspect regarding experimenting with measuring computations is that the assembly code produced requires an examination in order to check that the desired instructions are being performed. The default compilers' configurations remove code that seems useless, but which might correspond to the instructions under measurement.

To measure how the power measurements relate to the activity happening in the cores, a SNR is also to be calculated. For this, the measurements taken while the CPU is as idle as possible are used as noise and the noise plus signal values are measured while the computational load is at its highest value.

4.2 Sampling Method and Leakage Detection

In order to identify leakage, it first has to be defined. We define that some operation is leaking information through the energy registers if such information can be used to distinguish two similar operations using different data. In the context of cryptanalysis, it corresponds to distinguishing if the same algorithm used a different plaintext/key pair during two encryptions. This is the first step towards discovering sensitive information of the algorithm. In order to expose that leakage, it is possible to simplify the algorithm so that the leaking operations are not shadowed by the others. For the specific example of this case study using AES, in which the leaking part is the S-Box load in the *SubBytes*, different possible ways to simplify it include decreasing the number of rounds, reducing the variety of bytes in the plaintext and key, and limiting the available Hamming Weights of the loaded byte.

The low sampling rate available implies the need for a different approach than the ones used in other power analysis attacks, where an oscilloscope is used instead. Typical power analysis attacks, having a much higher samples-per-clock-cycle ratio (usually at least 1), power traces are acquired with a higher time resolution, that allows analysing and distinguishing different moments of the algorithm. In this case, a whole different situation happens since we do not have many samples per encrypted block. Actually, the reality is the opposite scenario, where there are several encrypted blocks than samples. This discards the correlation attacks that attack specific moments of the algorithm, since the time resolution is not available. On the contrary, a template-type of attack is a possibility, with some adaptations. The templates are constructed by repeating an encryption a large number of times with a key/plaintext pair that result in a specific Hamming Weight of the S-Box output at a certain key position, while acquiring the respective power consumption. To the distribution of that measurements, we call the template. These are used to match an attacking trace, which corresponds to the sampling performed in the machine under-attack while the cryptographic operations are occurring.

Following the previous definition of leakage, it proves necessary to distinguish between templates corresponding to different Hamming Weights. Being each template a distribution, statistical tests already exist to distinguish among them. The T-Test introduced in subsection 3.1.4 is an example, with the formulation at Equation (3.7) being suitable for data sets with possible different sample sizes and variances, such as the templates and the attacking traces. This way, the respective template to match with the trace will be the one that outputs a lowest T-value, meaning that it has the highest similarity.

4.3 Quantification of Leakage

To perform the quantification of the leakage the complexity of the scenario can be progressively increased in order to examine when the template distinction is no longer possible. In the case of AES encryptions under the simplified algorithm, this difficulty increase is achieved by increasing the number of rounds, possible Hamming Weights and variety of bytes towards the real case. This allows discovering how much of a real threat it represents.

4.4 Summary

This chapter presented the proposal method to develop in this work, with the goal of studying the threat of energy measurement capabilities of processors towards information security. The conceptual architecture was defined, as well as the methods to overcome some difficulties predicted, such as the lower sampling frequency.

Chapter 5

Using RAPL for sampling Power Consumption

In order to measure the energy consumption linked with the computed data at the cores, the domain under interest is the PP0, and is consequently used in all the energy acquisitions. The energy measures of this domain are stored in the 32-bit register *MSR_PP0_ENERGY_STATUS*. Recall that in side-channel attacks the leakage is done through the relative differences of the measures instead of their absolute value. In this work the energy values read will not be converted to Joules. Instead, the raw readings from the register are used. This explains the lack of a specific unit in some of the plots. Regarding the clock rate, the experiments are performed in a machine with a clock of 2.5GHz, and subsequent clock-cycles-to-time conversions were performed based on this specific value, unless otherwise stated.

5.1 Interfaces

To read the energy measurements stored by the CPU in the MSR, some interface must be used to provide such values to the user. The most common software used for this purpose is Intel's PAPI. However, given the existence of other options and the possibility to create a more specific and lighter interface, together with the requirement of acquiring the most accurate measurements, different interfaces are considered. This way, a comparison is made between three different choices: Intel's PAPI, the Power Capping Framework, and a framework created by other students and researchers of IST. The latter is designed as a Custom Framework.

Intel's PAPI - provides a ready-to-use generic interface that reads the MSR through the kernel module with the same name, and can also read from other performance registers. This results in a single, consistent and portable interface to perform power and performance measures, with the option of adding more components if desired.

Power Capping Framework (powercap) - is another framework with the same goal, that provides

the energy readings through the pseudo-file usually located at */sys/class/powercap/intel-rapl:0/energy_uj*. Unlike PAPI, powercap is used only for energy-related measurements. It is relevant that unlike PAPI, any user can read this file, without root privileges.

Custom Framework - was developed by IST students in order to reduce the overheads as much as possible. This way, it should be simpler and faster than the other options. For example, there's no need to be opening and refreshing a file constantly, like with *powercap*. Instead, a custom kernel module - the *MSRdrv* - is used together with a driver to directly read those values. It also provides the user with a better environment to run tests and simulations, by binding each thread to a core in order to reduce context switching, increasing the priority of the process, and clearing the CPUs pipeline before starting.

5.2 Sampling Rate

The sampling rate is one of the key-aspects of this work since that in contrast to usual power analysis attacks it's value is much lower than the CPU's clock frequency. It is therefore relevant to carefully specify it according to the framework. Another prime aspect to characterize is the Reading Frequency, that is the frequency at which the register is read. This reflects the time and computational power that the interface spends for reading the register, resulting in noise within the measures. It should be noted that the only way to know when the MSR is updated is to read it repeatedly, and between those updates the same value is provided. The algorithm 5.1 illustrates the method used for acquiring these values. The functions *readRegister* and *getTime* are different for each framework, as well as the set up of the environment and used variables, but all give the same output in the less computational time possible.

```
   input : S - Number of samples to capture
1  Eb = -1;
2  for n = 0 to S do
3  |  do
4  |  |  Ea = readRegister;
5  |  |  Reads++;
6  |  while Eb == Ea;
7  |  Intervals[n] = getTime;
8  |  Eb = Ea;
9  end
10 for n = 1 to S do
11 |  printToFile(Intervals[n] - Intervals[n-1]);
12 end
```

Figure 5.1: Sampling Rate and Reading Frequency Acquisition

Having the number of samples, reads, and time intervals between actual updates, one can compute the total time, t, by summing all those intervals. Then, the average sampling rate f_s and reading frequency f_r can be computed with

$$f_s = \frac{samples}{t} \qquad f_r = \frac{reads}{t}. \tag{5.1}$$

The figures 5.2, 5.3 and 5.4 show the histograms of the sampling intervals for each interface.

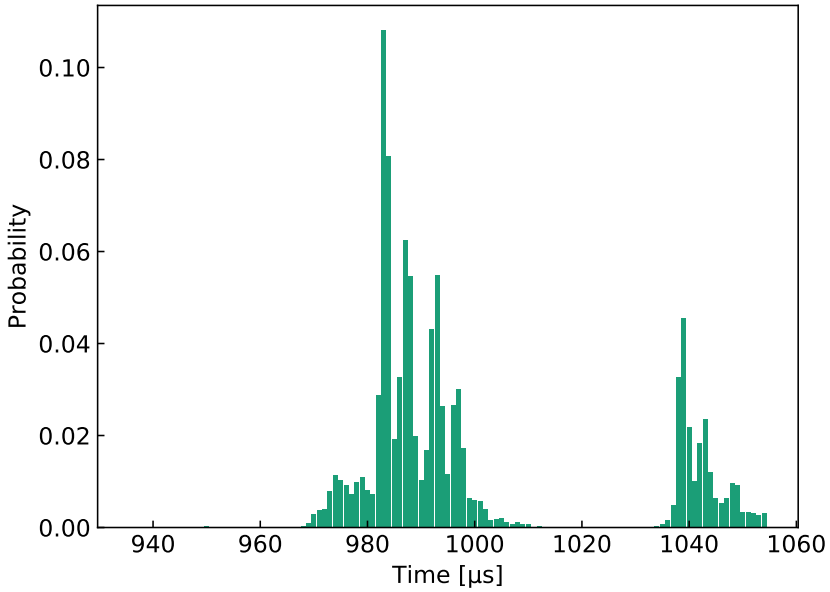

Figure 5.2: Sampling Interval Histogram of PAPI Framework. Mean Sampling Rate of 1.000 kHz and Reading Frequency of 575.35 kHz

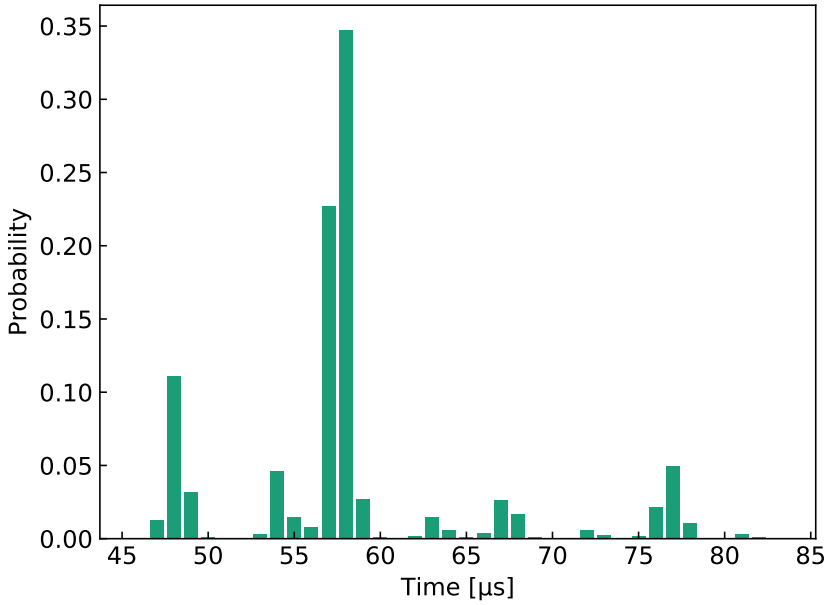

Figure 5.3: Sampling Interval Histogram of Powercap Framework. Mean Sampling Rate of 17.09 kHz and Reading Frequency of 103.79 kHz

One immediate aspect that stands out is the difference between reading and sampling frequency between PAPI and the other two frameworks, having an order of magnitude of difference. Several works and the very RAPL's guide mention that the update interval of the MSR is about 1 ms (frequency of 1 kHz) [32][13], but that value is in fact capped by PAPI, or relative to older architectures. The other two frameworks show a mean sampling interval in the order of 60 µs (frequency of around 17 kHz). This means that despite the register updating much more often, PAPI gives to the user only a portion of those

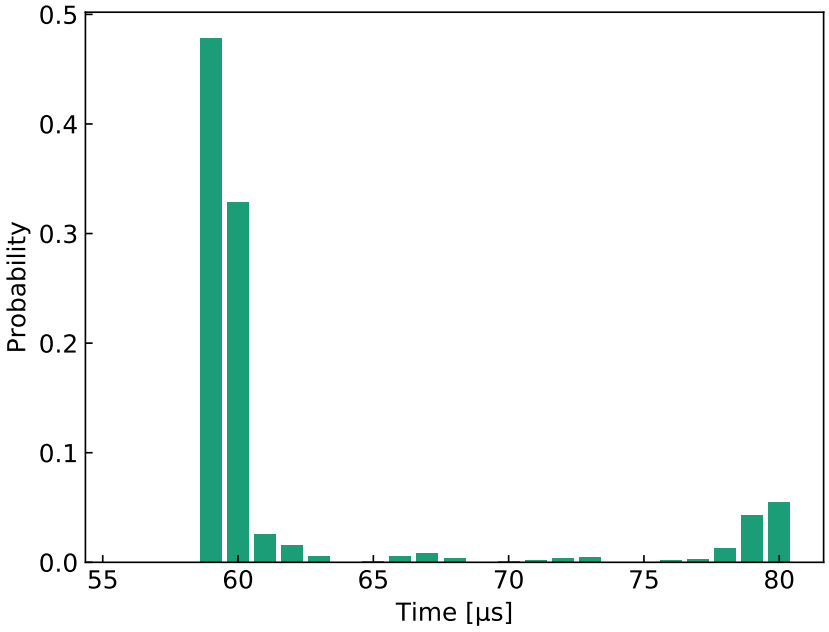

Figure 5.4: Sampling Interval Histogram of Custom Framework. Mean Sampling Rate of 17.22 kHz and Reading Frequency of 1033.12 kHz

updates.

Regarding the reading frequency values, the results were 575.35 kHz for PAPI, 103.79 kHz for Powercap and 1033.12 kHz for the Custom Framework. Here the significantly low overheads of the Custom Framework stand out, as opposing to the slow reading time of the Powercap. This result is expectable, as the latter framework reads the register through a text file, opening and closing it at each iteration, and such operations are significantly time-consuming. On the other hand, the Custom Framework uses its own driver to directly read the MSR, allowing faster readings.

A final relevant characteristic is that the sampling interval is not fixed. One could argue that this might be because of the framework's overheads, but the high reading frequency would not permit such a big disparity. Also, during an eventual attack this type of jitter cannot be avoided. Comparing the Powercap and the Custom Frameworks, the latter shows more promising results due to the higher reading frequency and sampling rate. Therefore, this framework will be used during the course of this research.

5.3 Influence of the Clock on the Sampling Frequency

A question that arises when having a low sampling-frequency-to-clock ratio is if it would be possible in an attack to lower the clock frequency in order to compensate that ratio. Different tools exist to easily lower the clock frequency, and even the BIOS can be used for that purpose. For the following experiments, the software used was *CPUFreq*. This gnome extension consists of a lightweight but powerful CPU scaling and monitoring system, which permits activating and disabling cores and changing their frequency as desired, through the kernel module with the same name. Because of these extra modules the resulting values (sampling and reading frequencies) are slightly lower than the previous ones, in a non-significant

way.

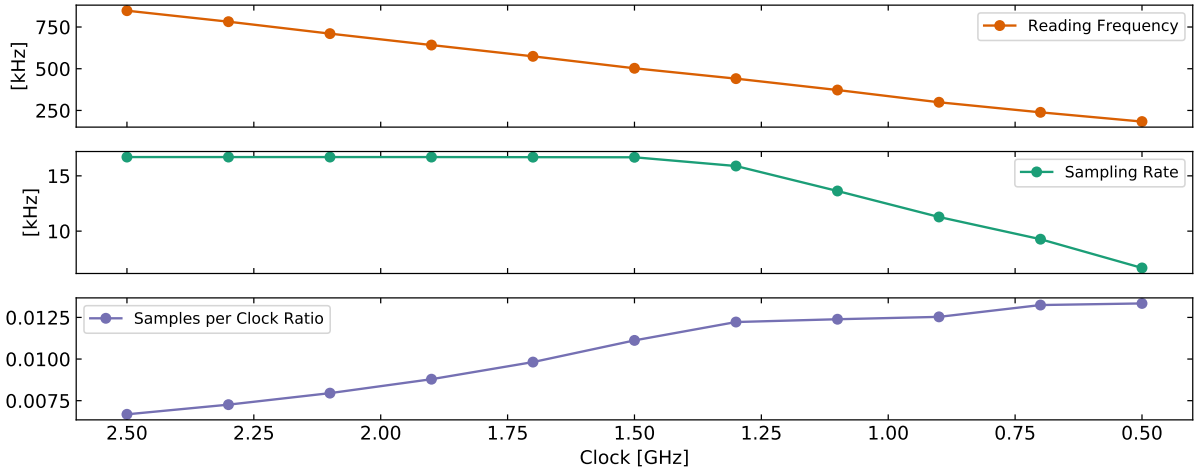

Figure 5.5: Influence of the clock on the reading frequency, sampling rate, and resulting samples per clock ratio.

Figure 5.5 shows the evolution of the sampling rate and reading frequency as the clock frequency decreases. The values are as expected, with the reading frequency gradually declining, starting to affect the sampling rate when the clock frequency hits 1.2 GHz. The bottom plot shows the samples per clock ratio, that is the number of samples that are acquired per clock cycle. It is possible to see that although this value rises, it converges at about 0.0125 due to the decrease of the sampling rate. Despite being higher than the original 0.007 at a full clock speed, it is still significantly below the desired 1. From this, it can be concluded that decreasing the clock frequency to achieve better samples per clock cycle is not a practical option, and a different way to face this challenge is required.

5.4 Idle Noise

The idle noise of this ADC is the result of measuring the computer energy or power consumption when minimising the existing processes and CPU usage. In fact, being part of a computer that hosts an operative system, it is impossible to eliminate all side tasks. Some system calls and processes will remain, which will eventually also take place during an encryption operation. Apart from those, the measuring thread also contributes to the consumed energy.

Energy and power measures are obtained in order to compare possible differences between them. Inspecting the driver and the RAPL guides, it can be seen that the power values are calculated from the energy and clocks measures, as $Power = Energy/T$. As seen before, the sampling rate and reading interval are not perfectly periodic, and this way the power measures will have some noise as a consequence of it. This affects the energy measures in the same way, making them similar, apart from the temporal scalar. During the acquisitions a reading interval of 30 µs is used in order to catch all register updates.

The graphs in Figure 5.6 show the almost identical idle energy and power plots, different mainly in the vertical scale.

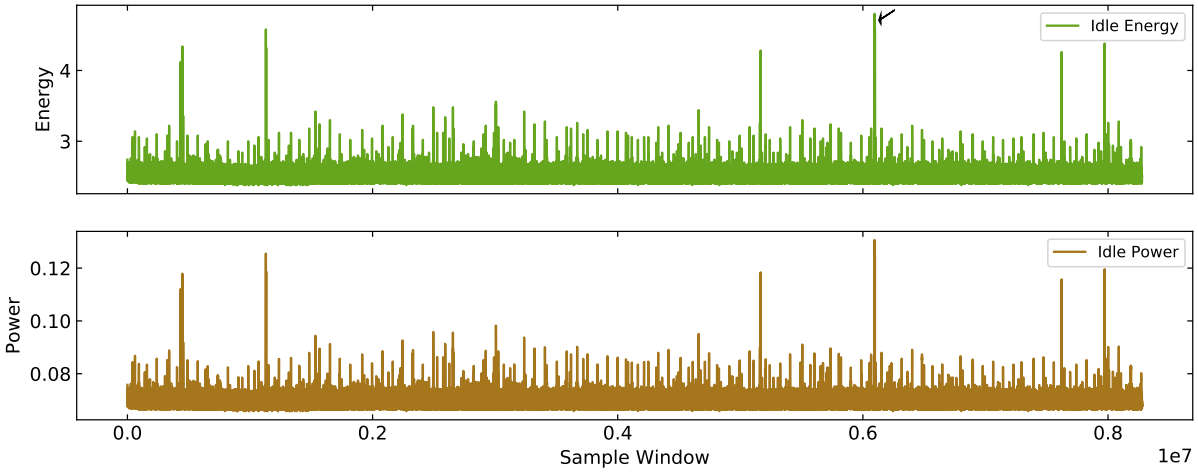

Figure 5.6: Moving average of idle energy and power during 5 minutes, with an average window of 50 samples. Idle acquisitions are done minimizing all running processes excepting the thread that reads the register.

An aspect from this figure that stands out is the existence of spikes. In order to find out why those happen, a zoom is applied to the graphic of the raw samples (instead of moving average) in the position of those spikes. This is performed in Figure 5.7 to the spike of the upper plot pointed by the arrow.

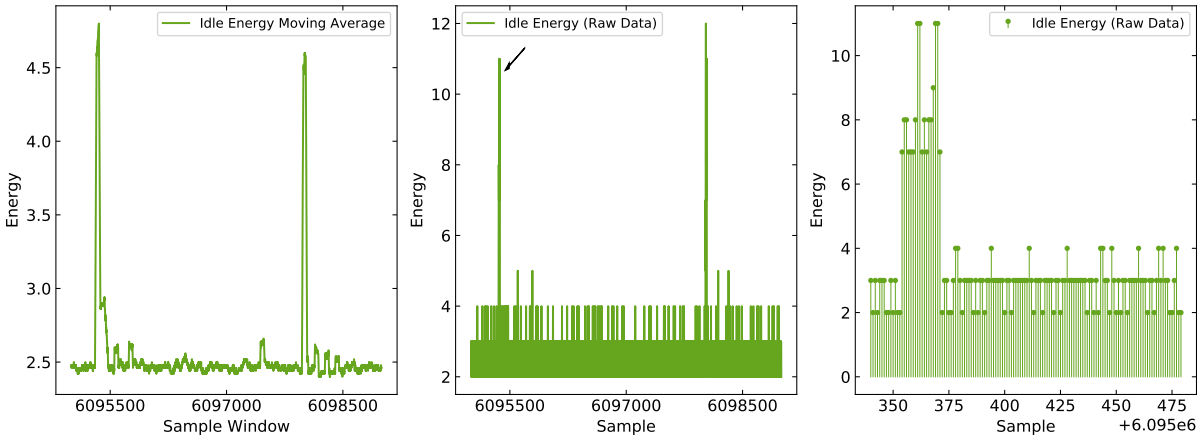

Figure 5.7: Different zooms of the pointed energy spike of the Figure 5.6. On the left, there is a direct zoom of the moving average, with a window of 50 samples. The middle graph shows the raw samples of that same zone. On the right a zoom is done on the pointed spike of the graph in the middle.

The left plot shows that in fact that spike is two spikes very close. Focusing on the left one, and viewing the individual samples, on the right plot one can see that the spikes are caused by consecutive high samples. Those consecutive samples are amplified in the moving average plot, appearing to be a much higher value. It is difficult to track the exact reason of this spikes happening. This experience was repeated, and the positions of the spikes shifted, which leads to believe this is due to system calls and unavoidable for now. This way, it is better to deal with them in post-processing than to prevent them from occurring.

There are three possible ways to solve the occurrence of these spikes. First, one can simply filter them out, by removing those samples from the data-set. Secondly, another option is to saturate them,

imposing a limit on the value they can take. A final and more complex option is to detect them and take it into account in the following calculations. In this work we use the first option, leaving the others to future research.

While this short-duration idle noise plot reveals some revealing properties of this ADC, it is also interesting to study a longer-duration acquisition to determine if the time difference between a possible template building and trace acquisition is relevant or not.

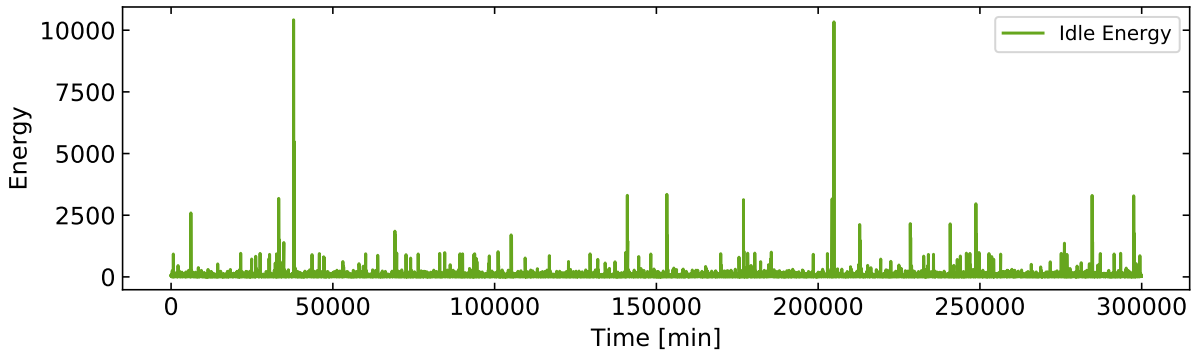

Figure 5.8: Idle acquisition during 500 minutes (approximately 8 hours) with register reads every 0.1 second resulting in reading sums of around 1666 register updates. This results in the higher energy scale. While the average consumption is constant, there are significant spikes.

Looking at the long term idle noise illustrated in 5.8, it can be concluded that the average consumption is constant, but some higher and unexpected spikes occur. This idle noise can be used to calculate the SNR of this ADC. For that, another acquisition is done with the CPU under full load, leaving all other conditions untouched using the SNR defined at Equation (2.2.4). This calculates what proportion of the output signal comes from background noise compared to how much comes from the actual computations of the process in question. This results in a SNR of 21.98 dB, which is considerably low for power measuring, but still allows for the potential existence of leakage. The method used to achieve a full CPU load was the *dd* command. This command is used to copy to and from files, and can therefore be used to shift a number of zero bytes in virtual memory by calling it with `/dev/zero` as source and `/dev/null` as destination.

5.5 Summary

In this chapter, a characterisation of RAPL sampling capabilities was performed with the goal of optimising its acquisitions in the next sections. The available interfaces between the hardware counters and the user were compared, and the *Custom Framework* showed the more promising results in terms of sampling and reading rates. Then, experiments were conducted that invalidated the possibility of lowering the clock rate as an attempt to improve the samples per clock ratio. Finally, the idle noise was defined and measured in order to calculate the Signal-to-Noise Ratio of this ADC. This allows understanding how much the encryptions affect the power consumption and the respective readings from the energy counters.

Chapter 6

Leakage Identification and Processing of Energy Samples

Having characterised the energy measurement tool, the goal of this chapter is to define the best method and parameters to capture power consumption information that can be used to detect the leakage. Furthermore, it is pretended to create templates and traces using the acquired profiles to break a simplified version of AES through some power analysis technique.

6.1 Simplified AES

In order to discover the best characteristics of energy measurement for power analysis (acquiring the samples to create the templates), a simple scenario is first defined that allows trying and evaluating different methods and parameters. This scenario is designed to permit gradual increase in complexity towards a real-case environment. The conditions are as follow:

1. 1 round of AES (instead of 10) - Allows the exposure of the leakage of the first S-box without the further rounds hiding it. Increasing difficulty can be achieved by adding more rounds.

2. All 16 bytes of the keys and plaintexts are the same - This amplifies the leakage of the S-box by 16. Gradual complexity added by using different values for part of the 16 bytes.

3. Lower number of possible Hamming weights of the values loaded from the S-box. In the limit it consists of having only two possible Hamming weights, 0 and 8, corresponding to the values `0x00` and `0xFF`, for the output of every S-box. By looking at the S-box table, the key/plaintext pair can be chosen to produce such a result. Difficulty increase can be achieved by adding the other possible Hamming weights.

Consulting the S-box output table, it's concluded that the values that load `0x00` and `0xFF` are `0x52` and `0x7D`, correspondingly. This leaves open the question of how to choose the plaintext/key pairs to

produce those values. The simplest way is to fix the key k at a value, for example `0x50`, and change the plaintext p such that $p \oplus \texttt{0x50} = \texttt{0x00}$ and $p \oplus \texttt{0x50} = \texttt{0xFF}$. This results in $p_1 = \texttt{0x02}$ and $p_2 = \texttt{0x2D}$.

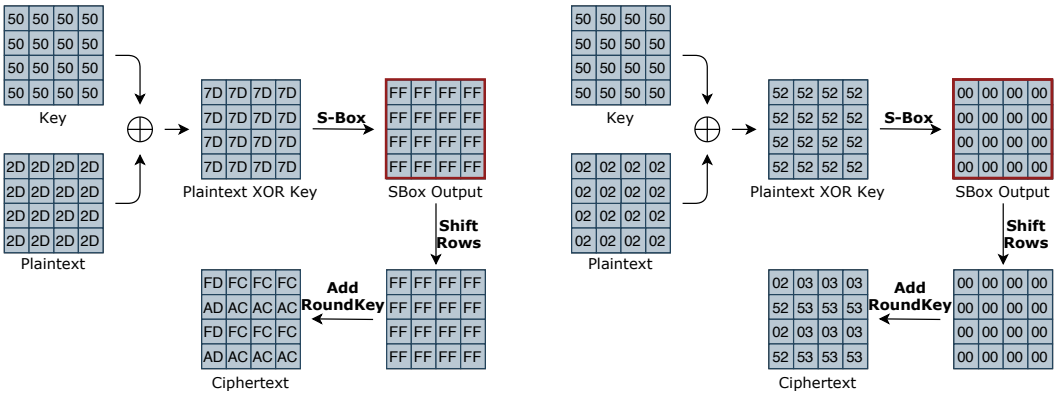

Figure 6.1: Simplified AES - Equivalent of the real AES beginning plus its last round, removing the rest of the loop. The chosen key/plaintext pairs result in a S-box output of full `0xFF`'s (left) and full `0x00`'s (right).

With the goal of obtaining the testing data to create the templates, the power consumption profile of a loop of encryptions are acquired under the simplified scenario conditions, with the chosen keys and plaintext such that the values loaded from the S-boxes differ. In order to produce the best results, the two plaintexts should be alternating after some amount of samples are acquired. This way, outside effects such as the room or computer temperature, the usage of the cores, or any other noise, like some temporary running process, affects in a similar way the various acquisitions. The amount of samples required is defined by the parameter *samples* in the Figure 6.2. Because everything happens in a single thread, the encryption loop has to be interrupted to read the register. The *iterations* parameter defines how many encryptions should be made before the register is measured. This value takes into account the time that each encryption takes, T_e (which changes from real scenarios to simplified scenarios because of the reduced number of rounds), and the desired frequency of reading the register, f_r. The time of reading the register is ignored because it is insignificant.

$$f_r = \frac{1}{iterations \times T_e} \tag{6.1}$$

Combined with the *samples* parameter, the duration of the acquisitions with each plaintext is defined. High duration of each acquisition, in the order of 60 minutes, lead to possible undesired differences in the values related to the noise described above. As the duration decreases, those noisy differences seem to disappear. The most promising results were found using values of around 5 minutes. The algorithm 6.2 illustrates this process.

A relevant choice of the acquisition is whether it is better to use the measured energy values directly or dividing them by the time intervals to calculate the power values. It was seen before that the results are almost identical, so it is not expected to make much difference. Recall that the plots do not display an unit because they represent the raw values retrieved from the register.

Looking at the graphs depicted in Figure 6.3 one can compare the moving averages of the consumed

input : Key, Plaintext-00, Plaintext-FF, iterations, samples
output: files with energy samples

```
key = KeyExpansion(Key);
while True do
    plaintext = NextPlaintext(Plaintext-00, Plaintext-FF);
    file = NextFile();
    for i = 0 to samples do
        for 1 to iterations do
            AES_Encrypt(key, plaintext, output);
        end
        Data[i] = ReadRegister();
        Time[i] = getTime();
    end
    PrintDifferences(Data, Time, file);
end
```

Figure 6.2: Acquisition Method - `0x00`'s vs `0xFF`'s

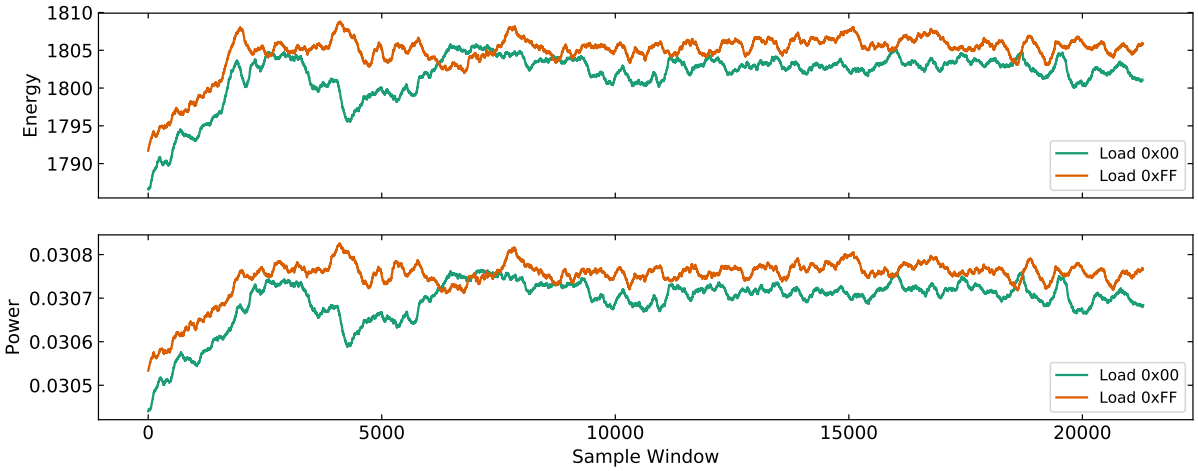

Figure 6.3: Power and Energy Moving Averages of loading `0x00`'s vs loading `0xFF`'s. Window of 55k (non-grouped) samples. It's visible that one operation consumes more energy than the other. Energy and power have similar plots.

energy and the power consumption while repeatedly loading `0x00`'s or `0xFF`'s from the S-boxes, in the simplified scenario, during approximately 5 minutes.

One aspect that arises is that the energy resolution is very high for the signals being acquired, resulting in a low granularity. Doing acquisitions with the minimum sampling interval possible (around 60 µs) and a resolution of 61 µJ results in the measured values being low and belonging to a small group of discrete values. In order to increase this resolution, some post-processing techniques can be used. One such technique consists of grouping by summing (or averaging) δ consecutive samples. The number of available samples decreases by δ, but the values have increased resolution. This happens due to the fact that the mean of a group of values can be different than any of these values. While this does not necessarily increase the chances of exploiting the measurements, it facilitates the filtering of outliers and allows for a better visualisation and demonstration of the leakage.

The histograms in Figure 6.4 show the corresponding distributions of the previous plots. The samples

are grouped in groups of 256 and filtered by three standard deviations from the mean to allow a better visualisation.

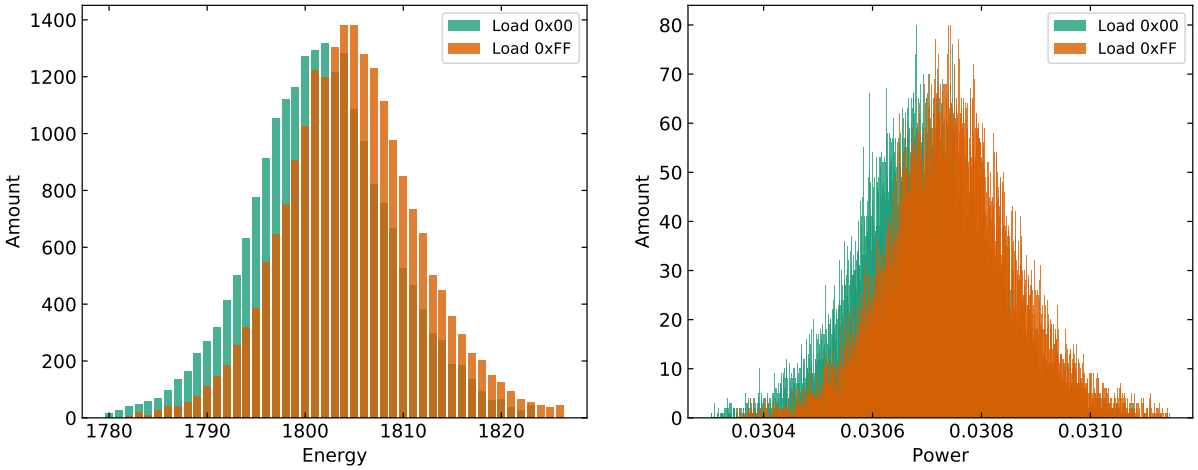

Figure 6.4: Power and Energy Histograms of Loading `0x00`s vs Loading `0xFF`s. The latter results in a distribution with a higher mean, illustrated in both plots. The differences in the x-axis result from the division of the energy by the time interval for the power histogram.

The figures demonstrate a clear difference of the energy spent when loading the different values from the S-boxes, with the loading of the bytes with value `0xFF` consuming more than the bytes with value `0x00`. The x-axis granularity difference of the histograms is explained by the division of the energy value by the time interval. While the energy values belong to a small set of discrete numbers retrieved from the MSR, the division of that number by a time interval - which can take a larger set of values - results in a much bigger group of decimal values. Another aspect that stands out is the increase of the plots in the beginning of the acquisition, which is studied in the following sections.

With this acquisition method it is possible to create a template for the operation of loading `0x00`'s and another for the operation of loading `0xFF`'s. Consequently, it is necessary to have a way to mathematically distinguish them in order to indicate which operation was performed based on the power consumption.

6.2 Template Matching and Differentiating

The desired function to match a trace to one of the possible templates takes as input a set of distributions (templates and trace), and returns values that reflect their similarity. The template that represents better similarity with the trace is the matched one. While this function's goal is to guess to which template the trace belongs, it can also be used to characterise and evaluate different parameters of the acquisition, in order to tune the most promising ones. This includes the usage of power versus energy values, the filters and grouping values to use, the and number of required samples for the templates and for the attacking traces.

To create this function, an operation is required that outputs the already mentioned similarity. One possible approach is to use the T-Test. Recall that the T-Test is a statistical tool that distinguishes

two data sets by evaluating their mean values, variances and sizes. Among the different formulations, the Welch's T-Test, formulated in Equation (3.7) has the advantage of permitting different sizes of the distributions, which is the case when a sample trace is corresponded to a template. This operation returns a value referred to as the t_{value}, that represents the similarity between those two data-sets, being zero when they are completely similar, and the absolute value increasing as the similarity drops. This value is usually compared against a critical value table (the T-Distribution table) to determine if the data-sets truly differ, or if that difference is due to the effect of chance. For the sake of simplicity, it is common practise to use the value of 4.5 to draw the line that distinguishes different sets from similar sets [33]. Because in this case we are interested in always selecting one template to match to the trace, the proposed matching function selects the template that together with the sample trace results in the lesser t_{value}, independently of being above or below that threshold. Future investigation can be done towards using a different matching function. For example, being a classification problem, machine learning algorithms could eventually be used for this task.

The size of the distributions is a relevant parameter for matching the traces to the templates. While the size of the template is limited by the available time and storage resources as the attacker computes them in a separate phase of the actual attack, a value should be defined for comparing purposes. On the contrary, the size of the attacking trace is a much different question, since in real cases it is usually not easily available. For example, it is normal to use a key only during a certain number of times before computing another one, or to only be able to encrypt during a certain amount of time on a compromised machine. It would not be suitable to consider an infinite number of samples for the attacking traces. This way, a maximum power trace size corresponding to 60 seconds of encryptions (approximately 1M samples) is considered given a clock frequency of 2.5GHz, that corresponds to approximately 360MB of encrypted data with the full AES (22.5k encryptions), or 4.2GB (262.5k encryptions) with the simplified scenario due to the reduced number of rounds.

For testing and comparing parameters the trace and templates are obtained from the same acquisition method. This way, the size of the template is the rest of the samples that do not belong to the trace. It should be noted that in a real attack this size can be much bigger, with that consequences being studied further.

One last requirement to evaluate the different parameters and conditions is a way to numerically measure the performance of that acquisition, based on the acquired samples. This performance reflects how well can the different templates be distinguished and each trace matched with the respective template, using the already mentioned T-Test method. The immediate thought, and the one used generally in cryptanalysis and side-channels, is to do the whole attack with those acquisitions and check if the guessed key is correct. To provide a continuous scale instead of a binary one, the position of the correct key in a ranking of keys can be used instead. This is supported by the fact that many cryptanalysis algorithms sort the possible keys by its probability of being the correct one, which allows discovering the real key in an intelligent and feasible brute force search even if the correct guessed key is not the first in that ranking. CPA is an example where this works well. However, on our case in particular, the probability of finding the correct key strongly depends on the number of plaintexts used for encrypting,

as seen in the next chapter. This makes the position of the correct key in a complete attack not the best choice, because it would require making acquisitions with a large number of plaintexts for each key guess. What can be used instead, being a common practise in classification problems, is the *confusion matrix*. The confusion matrix is a square matrix that describes the performance of a classification model by displaying the probability of choosing each predicted class (template) according to each actual class (trace). This way, the position i, j has the probability of matching the trace j to the template i. The number of rows and columns is the number of classes, that in this case represents the number of different possible Hamming weights for classification.

Using the confusion matrix for performance control has two advantages, besides the simplicity of the displayed value. First, being an important and widely used tool, there are different parameters derived from it that in a single value reflect aspects of the classification associated with that model. Examples include the false positives and negatives, the precision and sensitivity, or more complex indexes such as the F1 score and the Matthews Correlation Coefficient (MCC), that reflect the overall performance of the classifier. Secondly, the relationship between the confusion matrix and the probability of finding the correct key according to the number of plaintexts can be formulated, underlining the effectiveness of this metric towards this type of attack.

	Template 0	Template 8
Trace 0	**2.135**	7.955
Trace 8	17.113	**1.468**

	Template 0	Template 8
Trace 0	**0.88**	0.12
Trace 8	0.00	**1.00**

Table 6.1: Examples of T-Test results (left) and confusion matrix (right) for templates 0 and 8.

To calculate the confusion matrix, suppose an acquisition test with two resulting distributions, one with the Hamming weight of the S-box output as 0, and the other as 8. For each distribution, the samples related to 1 minute are considered the attacking trace, and the remaining are considered the template. Performing the matching function with one of the two attacking traces results in two t-values, that represent the possibility of that trace belonging to either template 0 or to template 8. Doing it for the other trace creates another row of two more values, creating the matrix on Table 6.1 (left). Using different groups of samples for the attacking trace and for the template allows this experience to be repeated for statistical purposes. Assuming that the template with the smaller t-value of each trace is the matched one, the confusion matrix on Table 6.1 (right) is created, providing the mentioned statistical values for the probabilities of matching each template to each trace. Naturally, a matrix with no errors would be a diagonal of 1's. To measure the performance of the classifications, the already mentioned Matthews Correlation Coefficient is used, which is a widely used coefficient which takes into account true and false positives and negatives, and presents many advantages over the usual F1 score [34]. When used on binary classification, MCC outputs a value between -1 and 1. In multi-class classification such as the present one, the minimum is a value between -1 and 0, while the maximum is always 1. Lower values represent a bad performance, while a perfect classification results in a MCC value of 1. Its formula is defined in terms of a $K \times K$ confusion matrix C by

$$MCC = \frac{\sum_k \sum_l \sum_m C_{kk} C_{lm} - C_{kl} C_{mk}}{\sqrt{\sum_k (\sum_l C_{kl})(\sum_{k'|k' \neq k} \sum_{l'} C_{k'l'})} \sqrt{\sum_k (\sum_l C_{lk})(\sum_{k'|k' \neq k} \sum_{l'} C_{l'k'})}} \tag{6.2}$$

In the following chapter it will be seen that some error in the matrix still leads to a successful attack.

6.3 Power vs Energy

Having provided a method to match the traces to the templates and evaluate different scenarios, it is necessary to decide whether to attack with the energy values or with the power values provided by RAPL, as well as understanding the influence of the size of the attacking trace. It was seen before that the distributions are very similar apart from the scale, but the question of which is more efficient (if any) remains. To compare it, the same experiment is performed twice, first using the energy values and secondly using the power values of the same acquisition campaign: three templates are used to guess the Hamming weight of tree traces, each belonging to one of these templates, with Hamming weights 0, 4 and 8. The rest of the simplified scenario criteria and all the other parameters are kept equal. The use of three Hamming weights instead of two results in increasing the difficulty slightly, and more importantly lowering the probability of a random classification being correct (from 50% to 16.67%). The size of the sample trace is gradually increased in order to see the required samples to correctly match it to the templates, and seeing the differences between the two approaches: with energy or with power. To make it statistically relevant, the test is repeated on many traces and the percentage of correct matches is evaluated, as depicted in Figure 6.5.

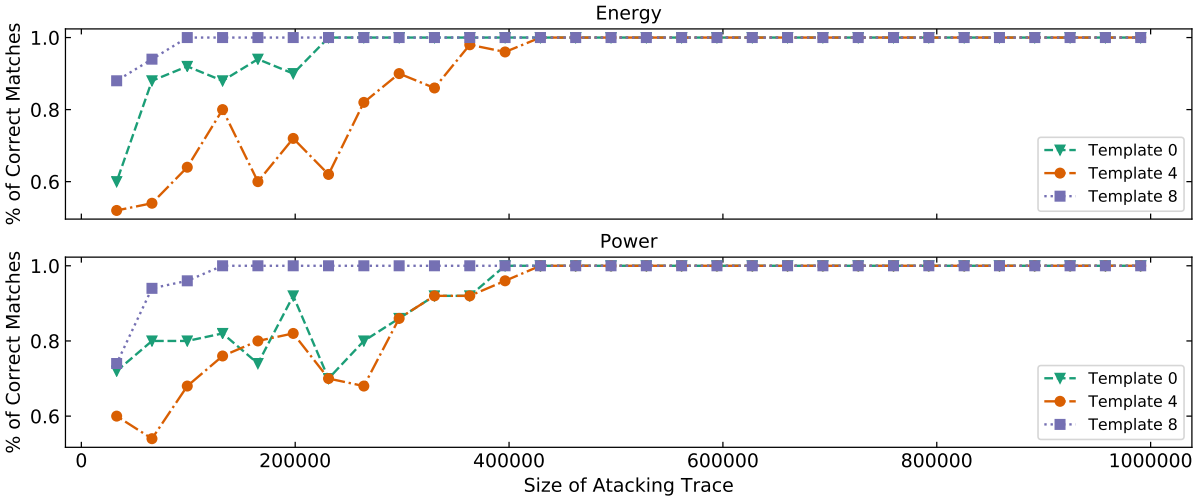

Figure 6.5: Probability of correctly matching the traces and the templates over the number of samples of the traces, for traces belonging to templates 0, 4 or 8. Comparison between using energy values (up) or power values (down). Using energy values has a slightly better performance where 100% accuracy is reached at approximately half a million samples.

It is possible to see that after around 0.5 million samples of the attacking trace, which corresponds to a trace captured during approximately 30 seconds, the template is correctly guessed 100% of the times if the hamming weight of the S-box output is either 0, 4 or 8, in the simplified scenario. This happens

despite choosing energy of power values. However, having to choose one, the energy values will be used for the rest of this study as it is easier to compute.

6.4 Outliers and Sampling Grouping

One important aspect in most distributions and acquisition campaigns is the definition of outliers and how to deal with them. As shown in Chapter 5, the idle noise contains some spikes corresponding to consecutive higher samples. These spikes can also be found on the acquired traces corresponding to the encryptions, and its importance or influence should be studied. As stated before, in this work they are simply eliminated from the data set. Different ideas exist for future work in order to avoid eliminating samples, which can be a non-optimal solution in real cases where the number of samples is limited.

To eliminate those and possible other outliers from the data-set, a filter is chosen. However, as seen in the previous sections, the raw data values are very similar, and therefore the filter has to be considered after grouping the samples to form a normal distribution. This way, the difference between using raw samples versus grouped samples should be compared, and consequently the influence of the filters. A common filter to eliminate outliers, and the one used here, consists of removing the samples at a distance of more than 3 standard deviations from the mean value.

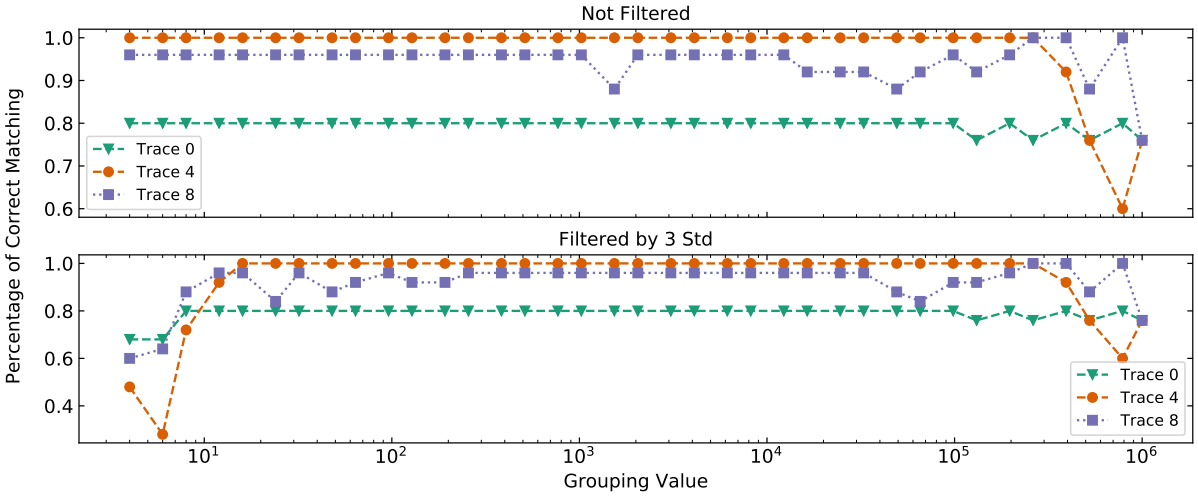

Figure 6.6: Comparison of the probability of correct matching each trace to the corresponding template for each different grouping values. The experiment is done without any filters, on top, and filtering the grouped values by 3 standard deviations from the mean, at the bottom. Results are similar across grouping values except in the extremes where the performance decreases.

In the values depicted in Figure 6.6 different grouping values are applied to the same data-set in order to study its effect on the performance of correctly matching each trace to its template. Furthermore, the experiment is made twice, first with no filter and then with the already mentioned filter by 3 standard deviations. Starting with the grouping values, it can be seen that the influence is not significant, as long as the values in the extremes are not used. For example, in the case of the filtered experiment (bottom plot), grouping values below 100 show some decrease in performance. In the top extreme, both cases showed lower success of classification when the grouping value drastically increases. This most likely

comes as a result of the reduced number of points of each distribution, as the original values are grouped in larger sets. Regarding the filter, the difference was not significant and further research work can be done, including other types of filters. In terms of computational power required for the computations, it decreases exponentially as the grouping values increases. This way, during the rest of this research samples will be arranged in groups of 256 and the filter by three standard deviations will be applied.

Machine Warm Up

Another important source of invalid data is what is refereed as *Machine Warmup*. As seen in the previous moving average of Figure 6.3, the beginning of an acquisition tends to have its values decreased and gradually increasing to the established trend afterwards. After noticing that this happens due to an idle time between acquisitions, during which no measures were being acquired, more attention was given to this issue.

This effect is illustrated in Figure 6.7. One can see that the longer the thread is idle via a sleep function, the lowest the energy values become when it resumes computations. This leads to a longer time reaching the trend value, that is maintained from that point forward. It is important to underline that during the sleep time there are no energy recordings. This means that the expected plot would not show the low-energy moments at all. Also, the window of the moving average is small enough so that these drops are not justified by any contamination of a very low sample to the corresponding window. In order to reassure that, the first sample after the idle time is removed.

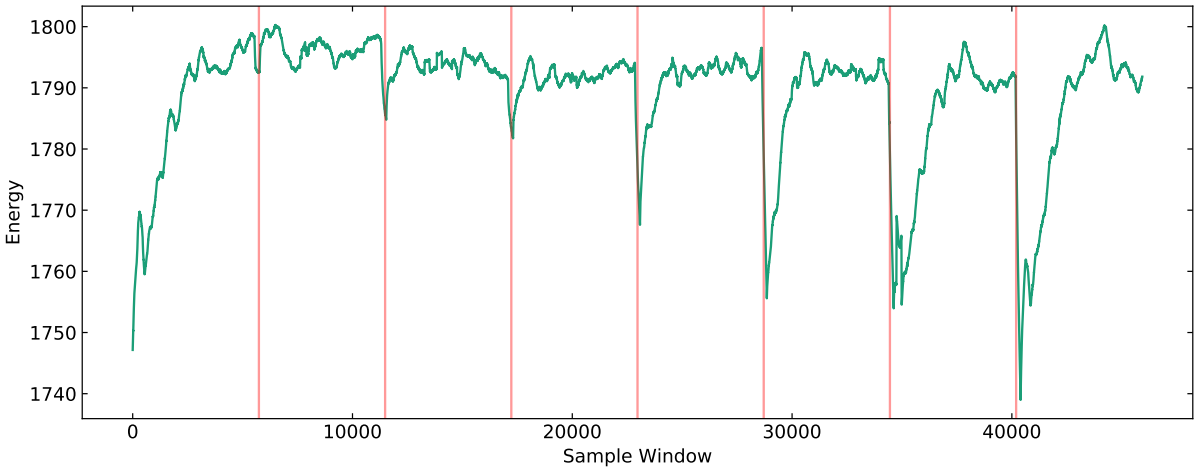

Figure 6.7: Machine Warm Up - the effect that the encryption loops seem to spend less power during the beginning, proportionally to the idle time before. Red bars represent idle moments.

The explanation of this effect is not found in literature, and can be further studied if future research. This way, the used method to tackle this effect was to avoid idle times between acquisitions and eliminate the first samples of every acquisition during post-processing.

6.5 Other Conditions

During the previous data acquisitions it was noted that while the results are reproducible, they are also somewhat noisy. That is, the same experiment can lead to slightly different outcomes, given the same parameters. In this section, we try to discover the conditions that result in those undesired differences. In addition to that, the previous analysis is done on another processor of the same family to reveal possible differences in the results. Discovering and evaluating such circumstances allows finding the most stable ones and getting the clearest acquisitions possible.

To compare among different situations, a number of acquisitions are obtained according to the algorithm depicted in Figure 6.2, using three Hamming weights, and the respective confusion matrices are calculated. To measure the classification performance described in each confusion matrix, the MCC introduced in Section 6.2 is used.

Multiple programs, priorities and recovery mode

The first proposed test focuses on the influence of having other programs in executing while the acquisition is occurring. While it is expected that this leads to higher power consumption, it is not clear how it affects distinguishing operations. Also, because of the limited number of cores, a higher number of processes does not necessarily mean a higher number of computations per time unit, as it will saturate at a certain value. On the other hand, if a parallel process consumes power in a constant way, its presence might not affect differentiating the used key during an encryption running in another process. Increasing the number of parallel processes and computational load can be done with the *dd* command in a Linux system, introduced in Chapter 5. On the contrary, in order to lower the number of side-tasks being executed it is possible to boot in recovery mode. This prevents many unnecessary background processes of being created, such as the graphical user interface.

From Table 6.2 it can be concluded that running the experiments in Recovery Mode in order to minimize the amount of noise processes leads to the clearest results, evaluated by its MCC. However, other interesting results are shown by the other rows of that table. The first row shows that a big overload on the CPU increases the difficulty of correctly matching the templates. This is expected, as there will be a high number of context switches between all the concurrent processes, that consume power from the cores in a not-constant way. Moreover, the sampling and encrypting process will not be continuously running, introducing time differences in the measures, and making the number of encryptions per measure inconstant. On the other hand, the second row shows that if only one core is stressed with a process that affects the power consumption in a constant way, then the templates can be successfully matched. This illustrates the most plausible scenario, where the power trace is obtained while the other process is doing the encryption.

One final consequence of this analysis towards vulnerabilities introduced by RAPL is the possibility to distinguish characteristics of running processes based on their power consumption profile, as seen in figure 6.8. This highlights how the possible threats provided by the energy counters are not limited to cryptanalysis.

Test Conditions	Average MCC
All Cores Stressed	0.15783
One Core Stressed	0.48164
Recovery Mode	0.53033

Table 6.2: Influence of other processes in the acquisitions. *Stressed Cores* means the number of cores under heavy computational load. Being a machine with two cores, the results drastically decrease when more than one core is stressed, as constant context switching is necessary between the heavy load and the sampling thread.

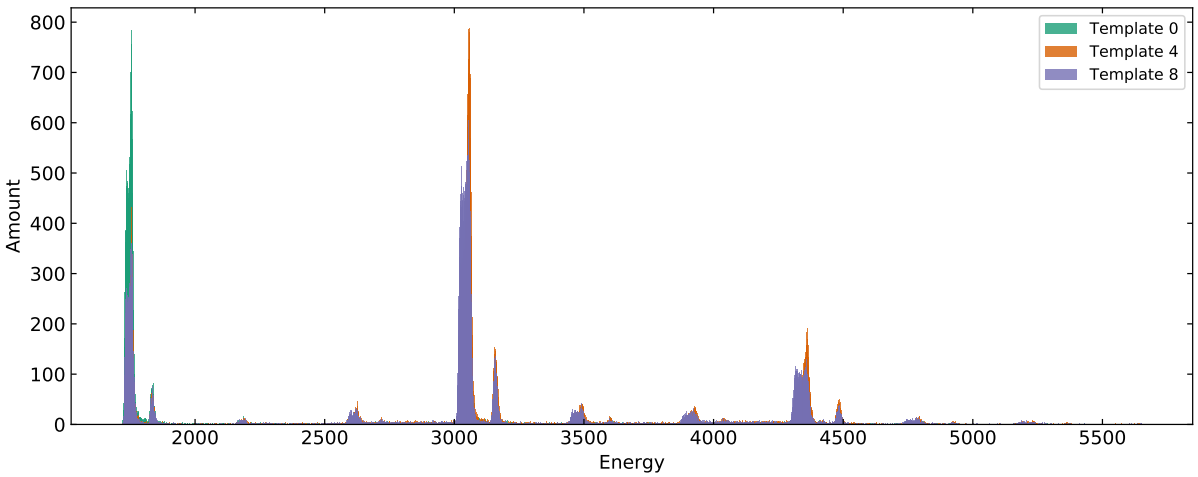

Figure 6.8: Energy acquisition with many different processes. Information regarding the processes and the CPU utilization is leaking.

Other architecture

To study the possible differences between RAPL across different processors, the same experiments done before are repeated in another machine. This time, a Intel i7-8700k with 6 cores and a base clock frequency of 3.7 GHz. Among all the differences among this processor and the previous one, the clock speed and the number of cores should be noted. Having an even higher clock speed, the number of samples per clock cycle studied in chapter 3 decreases even further. Perhaps more relevant is the increase in the number of cores, from 2 to 6, that comes with an overall increase in the energy consumption of this CPU.

The histograms of the sampling interval depicted in Figure 6.9 show that the RAPL behaviour is similar when using the same framework to acquire the values. The faster clock frequency resulted in a higher Reading Frequency of 1400.42 kHz (compared to the 1033.121 kHz of the previous machine) but the sampling rate is approximately the same, with a value of 17.336 kHz. The distribution of the values is slightly different.

To study the leakage in this machine the simplified scenario is used again, with the templates 0, 4 and 8. From the results illustrated in Figure 6.10, different observations are taken. First of all, there is a significant deviation from a normal curve happening in the three cases. While being difficult to track the exact explanation of this, it is most likely that it happens due to other processes of the system and not from the data manipulation itself. Secondly is the fact that unlike the previous results for the other

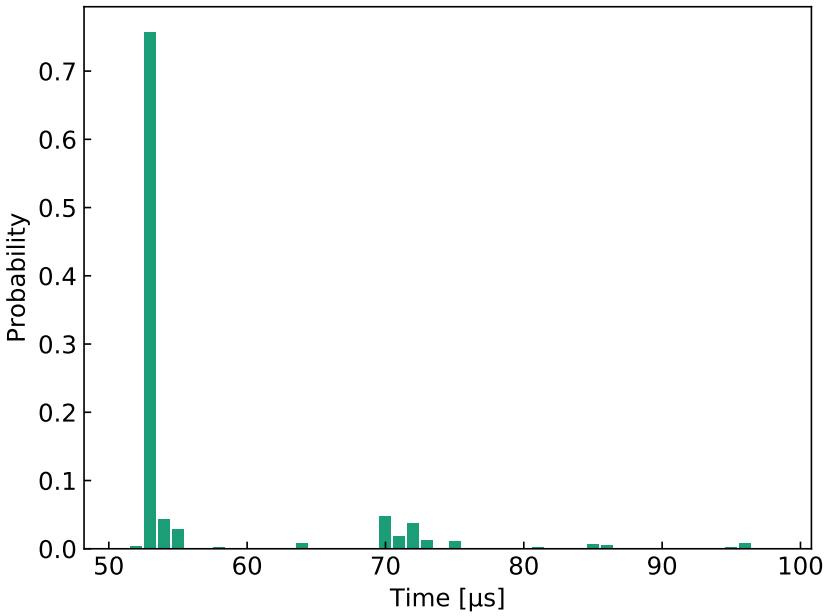

Figure 6.9: Sampling Interval Histogram of Custom Framework in a different CPU. Mean Sampling Rate of 17.336 kHz and Reading Frequency of 1400.42 kHz

machine, the histograms completely overlap each other. These results were repeated and this was observed every time. The proof of leakage does not happen so firmly in this case. The most probable reason is the increase of the number of cores, making the parallel computations of other processes overshadow the leakage resulting from the data processed in one of the cores.

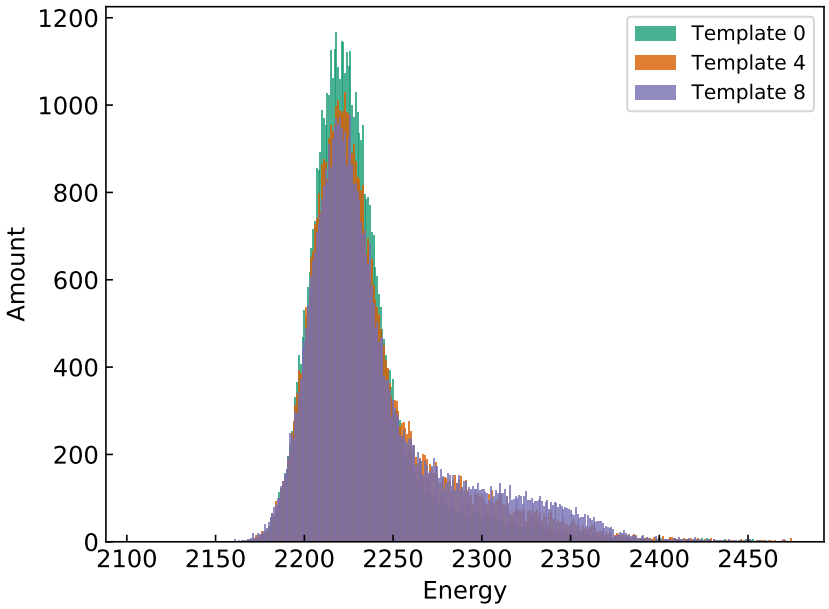

Figure 6.10: Histograms of simplified scenario for a different CPU. The results are substantially different, as the leakage is not visually detected in this case.

In terms of performance of the classifier, the resulting Confusion Matrix illustrated at Table 6.3 shows that two of the three traces were wrongly classified most of the times, resulting in a MCC of around 0,

that represents an almost random classification. This result, already expected by the plot of 6.10, shows how the leakage measured by interfaces like RAPL differs from machine to machine. This reflects how the targeted processor is an important feature of a possible attack, as they provide different levels of resistance against side-channel attacks.

	Template 0	Template 4	Template 8
Trace 0	**0.72**	0.1	0.18
Trace 4	0.30	0.24	**0.46**
Trace 8	**0.58**	0.20	0.22
MCC: 0.09479			

Table 6.3: Performance of classification using a different architecture. MCC is similar to a random classification.

6.6 Summary

This chapter starts with the description of the simplified version of AES used in a preliminary attack. Then, the algorithm for performing the acquisitions while encrypting was defined and used to reveal the existence of leakage. Following that, the challenge of the templates matching and differentiating was exposed together with the proposed solution. Consequent analysis was performed regarding the difference between the power and energy samples, and towards data processing and outliers removal, with its effectiveness towards templates differentiating in mind. Following that, other conditions were compared such as the load of the CPU with the goal of optimizing future template building. Finally, other machines were used to perform the same tests, in order to analyse the power measuring behaviour of different processors.

Chapter 7

Evaluation of Attacks and Potential Threat

7.1 Definition of Attacks

In this section an attack is designed to recover the secret key from AES using the last sections' template matching function and acquisition method. The goal is to study its potential towards a real-case scenario, starting from the simplified one. The algorithm described in 7.1 illustrates a complete attack, highlighting how matching templates to the Hamming weight of the S-box output can be used to recover the key during the *attacking phase*.

Input: HW - Possible Hamming Weights of Templates, P - Number of Plaintexts, K - Possible Keys

Profiling Phase :

1 **for** $h \in HW$ **do**
2 | Templates[h] = *createTemplate()*;
3 **end**

Capture Traces :

4 **for** $i \in 0, ..., P$ **do**
5 | Trace[i], Plaintext[i] = *Encrypt(plaintext = rand())*;
6 **end**

Attacking Phase:

7 **for** $i \in 0, ..., P$ **do**
8 | h = *MatchTemplates(Templates, Trace[i])*;
9 | **for** $k \in K$ **do**
10 | | **if** $HW(SBox(k \oplus Plaintext[i])) == h$ **then**
11 | | | score[k]++;
12 | | **end**
13 | **end**
14 **end**

Figure 7.1: Template Attack against AES

As stated in previous chapters, the first part of the attack consists of creating templates in the cloned

device owned by the attacker. On the second part, the attacking traces are captured on the machine under attack. Note that this can be done at any time, independently of creating the templates. The plaintexts used for encrypting with the unknown key are stored to be used later.

Finally there is the attacking phase, when the attacker tries to recover the secret key part from the h generated templates and the P pairs of captured traces and corresponding plaintexts. For that, the function *MatchTemplates* takes the templates and each attacking trace to guess h, that is the Hamming weight of the output of the AES S-box under attack. Knowing that value, one can iterate all the possible keys and calculate which of them would, *XORed* with the plaintext associated to that trace, load a value from the S-box with that same Hamming weight h. The score of such keys is incremented, and the loop continues for the next trace/plaintext pair.

At the end of that cycle, each key will have a score. Since the used key was the same for all the iterations, assuming that the matching function classifies correctly, there will be one possible key with the maximum score similar to the number of iterations, which is the correct key. The usage of a score allows for an intelligent brute force search in case the accuracy of the templates matching function is not 100% and the correct key is not the one with the highest score. In this cases, *brute forcing* the key using the higher-scored guesses before the lower-scored ones allows finding the correct key in a feasible time (depending on the actual score ranking of the correct key guess) as opposed to a normal brute force.

Other factor that can compensate for poorer classification accuracy is the number of available pairs of plaintexts/traces. This represents the number of power traces that the attacker is able to obtain from the targeted machine, and the corresponding plaintexts. In order to effectively study this happening, a simulation can be done with a function that mimics the template matching according to an error parameter, P_e, indicating the percentage of choosing a wrong template given a trace and the group of possible templates. This simulation is illustrated in Figure 7.2.

Input: T - Templates' Hamming Weights, P - Number of Plaintexts, P_e - Probability of choosing wrong template

```
1  Key = getRandomByte();
2  Plaintexts = getRandomPlaintexts(P);
3  for k ∈ 0, ..., 255 do
4  |  for p ∈ 0, ..., P do
5  |  |  h = simMatchTemplates(T, HW(SBox(Key ⊕ Plaintext[p])), P_e);
6  |  |  if HW(SBox(k ⊕ Plaintext[p])) == h then
7  |  |  |  score[k]++;
8  |  |  end
9  |  end
10 end
11 if argmax(scores) == Key then
12 |  Success!;
13 end
```

Figure 7.2: Simulated Attack

Notice that while it may seem possible to generate the plaintexts in a way more clever than randomly, the unknown key together with the non-linearity provided by the S-box makes it unfeasible. Also, given the constraints of the simplified scenario, the generated plaintexts have to result, after being *XORed*

with the key, in a value that loads from the S-box a byte with a valid Hamming weight. To understand the number of available plaintexts we recall the distribution of the Hamming weights of the numbers belonging to a byte:

- One value with Hamming weight 0 and 8 (`0x00` and `0xFF`),
- 8 values with Hamming weights 1 and 7,
- 28 values with Hamming weights 2 and 6,
- 56 values with Hamming weights 3 and 5,
- 70 values with Hamming weights 4.

This way, assuming the Hamming weights 0, 4, and 8, there are $1 + 1 + 70 = 72$ available plaintexts. The obtained results, depicted at Figure 7.3 result from repeating the simulation for a continuous number of plaintexts and values for the error parameter and evaluating the position of the correct key in the ranking of possible keys, based on it's score.

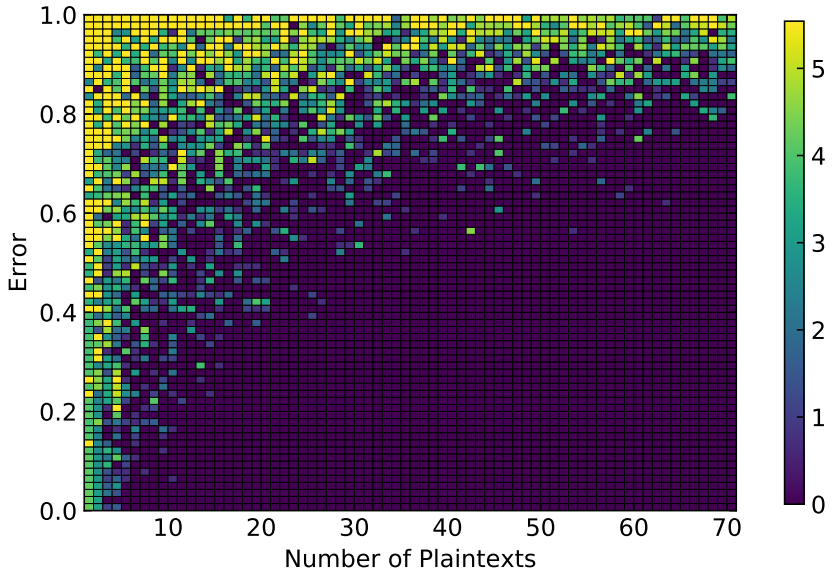

Figure 7.3: Logarithm of key ranking per error rate and number of plaintexts for templates with Hamming weight 0, 4 and 8. As the number of plaintexts increases and the error decreases, the key ranking decreases, meaning that the correct key is found. Even with high error values, it's possible to find the correct key given enough plaintexts/traces are provided. The logarithm is applied to better distinguish the results.

One can see that increasing the number of plaintexts leads to higher chances of finding the correct key, as the corresponding position is lower. Here, the ranking means the number of keys with a score higher or equal to the correct key. This way, a ranking of 0 means that the correct key is found. Despite the fact that lowering the error rate contributes to finding the correct key, it is relevant that such a key can be found even with significantly high error rates (higher than 0.5), given enough plaintexts. Since the error rate of the templates matching function were actually lower than that, it can be concluded that this attack works for the simplified scenario, using three possible Hamming weights, with a feasible number of plaintexts/traces.

7.2 Increasing Difficulty

After concluding that the simplest scenario (with the hamming weight 4 included) can be successfully attacked, the difficulty is increased in order to evaluate how much of a real threat this attack is, or how far it is from being one. For that, such difficulty is raised independently in three orthogonal ways: increasing the number of rounds, using keys with different bytes and adding different possible Hamming weight templates to match to the output of the S-box.

7.2.1 Adding Possible Hamming Weights

As more Hamming weights and the corresponding templates are included, it gets more difficult to match them. This happens firstly because there are more classifying options, and secondly because they are more similar to each other. In order to keep the biggest differentiation between the Hamming weights of the templates, we start by adding the weights 2 and 6.

	Template 0	Template 2	Template 4	Template 6	Template 8	Error
Trace 0	**0.54**	0.24	0.16	0	0.06	0.46
Trace 2	0.12	**0.62**	0.24	0.02	0	0.38
Trace 4	0.20	0.26	0.12	**0.42**	0	0.88
Trace 6	0	0.02	0.22	**0.76**	0	0.24
Trace 8	0	0.02	0	0	**0.98**	0.02

Table 7.1: Confusion matrix with 5 different Hamming eights. The probability of error increases while comparing to only 3 Hamming Weights, leading to an significant error in the classification of Trace 4.

Table 7.1 illustrates the confusion matrix of the matching function regarding the 5 indicated templates. As expected, the overall error rate is higher than with less Hamming weights, and the Trace 4 is on average wrongly classified as template 6. The results of Table 7.2, with all the nine Hamming weights, illustrate that the error rate is particularly high and most of the traces are wrongly classified.

	Temp 0	Temp 1	Temp 2	Temp 3	Temp 4	Temp 5	Temp 6	Temp 7	Temp 8	Error
Tr 0	0.16	**0.44**	0.14	0.04	0.10	0	0	0.06	0.06	0.84
Tr 1	**0.72**	0.14	0.10	0.02	0.02	0	0	0	0	0.86
Tr 2	0.08	0.04	**0.40**	0.28	0.14	0	0	0.06	0	0.60
Tr 3	0.12	0.16	0.18	0	**0.26**	0.06	0.02	0.20	0	1.00
Tr 4	0.14	0.06	0.18	0.06	0.08	0	0.20	**0.28**	0	0.92
Tr 5	0	0	0	0	0.16	0.12	0.02	**0.70**	0	0.88
Tr 6	0	0	0	0.06	0.08	0.14	0.02	**0.70**	0	0.98
Tr 7	0	0	0.14	0.08	0.18	0.10	0	**0.50**	0	0.50
Tr 8	0	0	0.02	0	0	0	0	0	**0.98**	0.02

Table 7.2: Confusion matrix with all Hamming Weights. Error rate is gradually increased, as only the Traces 2, 7 and 8 are on average correctly classified.

Again, the question that arises is at which extent does that error compromise the attack. The results of running the previous simulation for 5 and 9 templates are illustrated in 7.4. In the case of 5 templates,

with most errors between 0.2 and 0.5, around 10 plaintexts would be enough to correctly find the secret key, unless it belonged to the template 4, that had an error of 0.88. On the other hand, in the case of all 9 templates, the high error rate makes it very difficult to obtain the correct key this way, even with a non-realistic number of plaintexts.

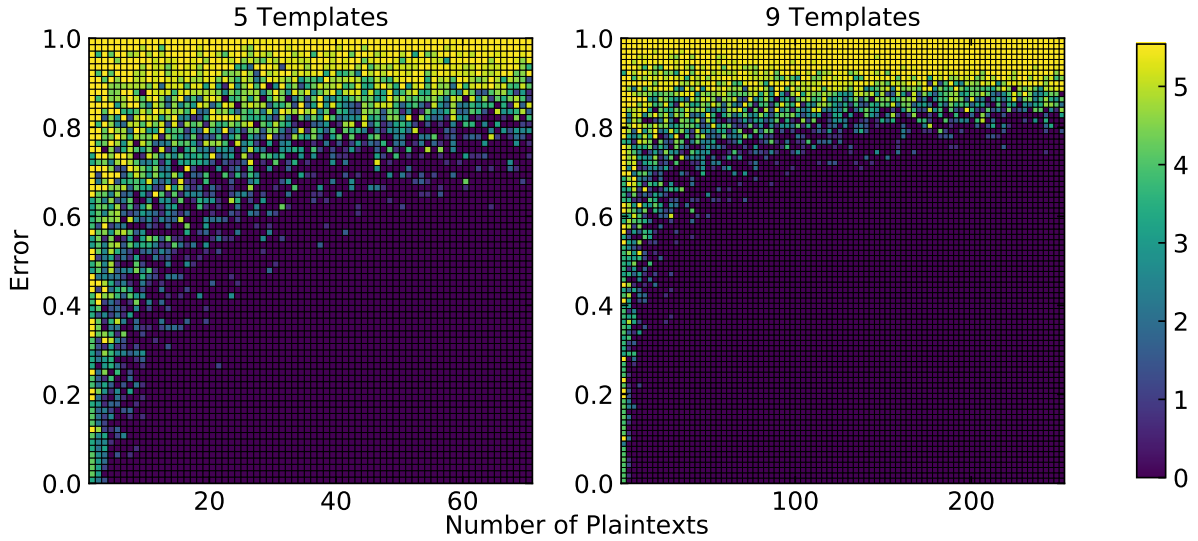

Figure 7.4: Logarithm of key ranking per error rate and number of plaintexts for templates with hamming weight 0, 2, 4, 6 and 8 (left) and all nine templates (right). The logarithm is applied to better distinguish the colors. The results are similar to figure 7.3 in the way that the key can be correctly found with a certain error rate, assuming enough plaintexts are provided. However, error rates higher than around 0.8 greatly compromise the attack, even with an unlimited number of plaintexts and traces.

While these difficulties compromise the attack against a system using the real AES, ideas exist to contour them. For example, it is possible to exploit the fact that the miss-classifications tend to hit a template with an adjacent Hamming weight. This proposal is out of the scope of this work and kept for future research.

7.2.2 Adding Different Bytes

In the simplified scenario, the keys and the plaintexts have all the 16 bytes alike. This means that the leakage in a S-box is being multiplied 16 times. Such does not happen in a real case. Besides that, it is necessary to isolate small parts of the key to make an attack feasible. The proposed solution is to randomise the outputs of some S-boxes, ideally all but one, so that the non-randomised one can be isolated. This is usual practise in cryptography as the random values tend to null each other with the increment of sample. Randomising the S-boxes output is easily achieved by randomising the correspondent part of the plaintexts.

One aspect worth mentioning is if it is mandatory to isolate the bytes completely, or if attacking groups of bytes (for example two) would be enough. Here, it is important to recall that breaking the key in different bytes is what permits this type of attack, because it leads to a search space sized $16 \times 2^8 = 4096$ instead of the impossible 2^{128}. Attacking the bytes in pairs results in a search space of $8 \times 2^{16} = 524238$ which while still possible, highlights how fast it grows. However, the number of required

plaintexts/traces for such an attack would make it infeasible. This way, it is fundamental to be able to distinguish a singular byte.

The idea here is to study how the number of randomised bytes influence the leakage, and eventually if it is possible to isolate one byte at a time.

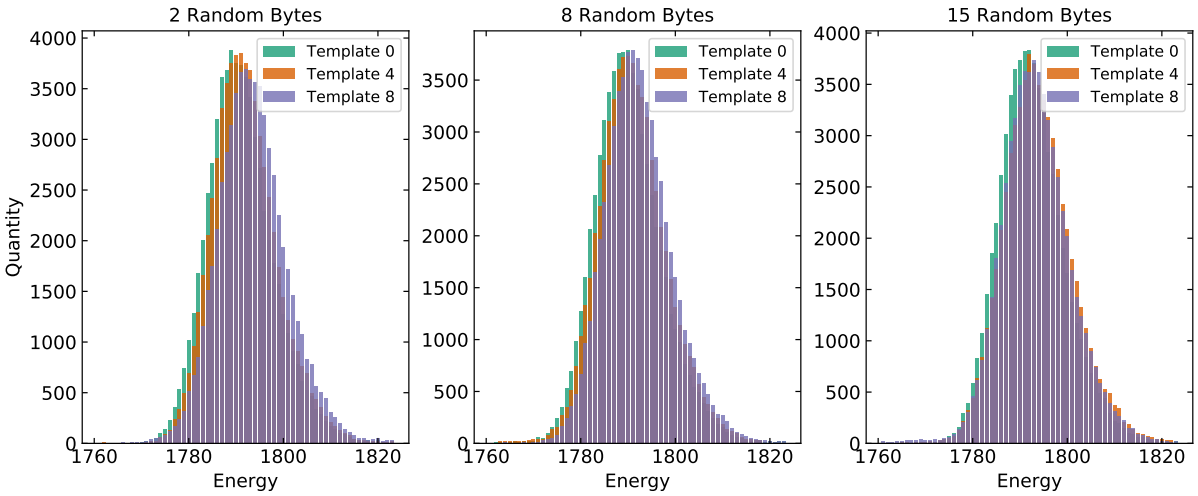

Figure 7.5: Distributions with 2, 8, and 15 random s-box inputs/outputs. This leads to isolating 14, 8 and 1 bytes. As the number of random bytes increases, they become less distinguishable.

The graphs in Figure 7.5 show how increasing the number of random bytes makes the distributions less distinguishable. While with 8 random bytes it is still possible to visually see some differences, at 12 random bytes the two histograms completely overlap each other.

2 Random Bytes			
	T0	T4	T8
Tr 0	**0.62**	0.24	0.14
Tr 4	**0.38**	0.28	0.34
Tr 8	0.16	0.16	**0.68**
MCC: 0.29378			

8 Random Bytes			
	T0	T4	T8
Tr 0	**0.70**	0.24	0.06
Tr 4	**0.54**	0.16	0.3
Tr 8	0.06	0.38	**0.56**
MCC: 0.21258			

15 Random Bytes			
	T0	T4	T8
Tr 0	**0.74**	0.1	0.16
Tr 4	**0.48**	0.34	0.18
Tr 8	**0.54**	0.34	0.12
MCC: 0.10865			

Table 7.3: Confusion matrices for distributions with 2, 8 and 15 random s-box inputs/outputs. Higher random bytes lead to lower classification performance.

In terms of influencing the attacking capabilities, the resulting confusion matrices of each case, illustrated in Table 7.3, show that as expected, trying to isolate less bytes by randomising the remaining ones results in a decrease of classification performance.

7.2.3 Increasing Number of Rounds

By increasing the number of rounds, the leakage at the S-boxes of the first round becomes overshadowed by the other rounds. Recall that unlike usual attacks, the time resolution of the acquisitions does not permit isolating certain parts of the trace. However, since the rows and columns are mixed in the rounds of AES, the randomised bytes explained before will create whole random states after the first

round. Again, in theory this random values will become null as the number of samples increases, being the only non-random component the S-box operation of the first round, which leaks the key information.

Figure 7.6 illustrate the difference of the two distributions, loading `0x00` and loading `0xFF`, as the number of rounds increase, with the rest of the simplified scenario criteria. The differences in the energy axis between plots are a consequence of the number of repetitions per sample, that changes according to the number of rounds. Similar to increasing the number of random bytes, the distributions become less distinguishable and the classification performance decreases.

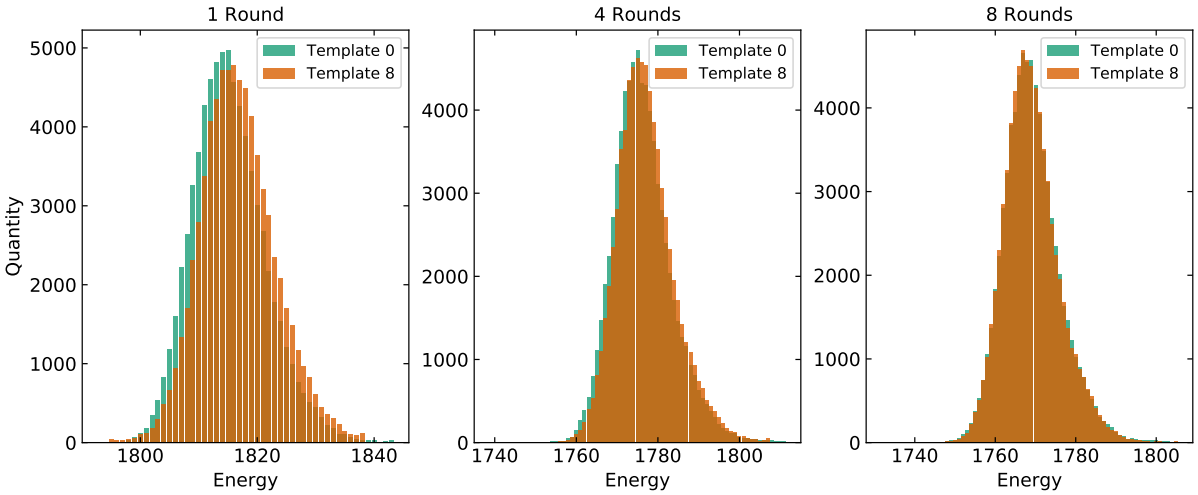

Figure 7.6: Distributions with 0, 4, and 8 rounds. As the number of rounds increases, they become less distinguishable.

Together with the difficulty added when increasing the number of randomised bytes, the attack becomes compromised by the necessity to isolate both bytes and rounds. While it might be possible to find some leakage regarding a single byte in the first round with the current method and setup, it would require an unrealistic number of encryptions for a real-case scenario.

7.3 Summary

In this chapter the main goal was to create an attack and study its performance against the simplified AES defined before, and then against different versions closer to the real AES. This was done independently by adding possible Hamming Weights to the templates and key choices, by adding different bytes to the key and plaintext and by increasing the number of rounds. Parallel to this, a simulation method was defined to study the soundness of the attack and its effectiveness when using different plaintexts, for various conditions. The attack successfully discovers the key in simplified scenarios, but is not successful in the real-case, as the increased number of rounds and possible Hamming Weights together with the necessity of isolating each byte separately made the error rate of the template matching function rise too much.

Chapter 8

Conclusions and Future Work

Information security is an extremely important aspect of today's modern society, as the digitalization and information technologies have an ever-increasing importance in most economic and social systems. The constant evolution of technology and creation of new features result in safer and improved solutions, but also bring new challenges and vulnerabilities that can be exploited. Extreme precaution must be taken, specially for widely-used and trusted devices such as computer's processors, namely Intel's. Side-channel attacks in particular require an extra attention given their difficulty to detect and prevent, in conjunction with their dangerous potential.

This research builds a foundation to the study of software-based energy measuring side-channel attacks against AES, in an attempt to completely recover the secret key from the power consumption profiles. To start, the characterization of RAPL as an ADC was performed, including the comparison of different interfaces. It turned out that the most-commonly used interface, PAPI, is not the best for fast acquisition, since it limits the information provided by the RAPL registers.

Following that, a simplified scenario was designed in order to perform experimental work, creating the basis for gradual attacks aiming the Advanced Encryption Standard, which is the must used symmetrical encryption algorithm. Analysing the power traces of encryptions under such conditions concluded that there is a leakage that can be detected and possibly exploited. That is, the relationship between the processed data and the energy or power consumption can be detected with RAPL, in a way that can compromise it's security, causing a threat for sensitive information. A method was created to compare the power profiles acquired during the encryption operations with previously done ones, recalling the template attacks. This allowed discovering the Hamming weight of the value loaded from the first S-box during the encryptions. From this result, an algorithm was designed to use it together with the original plaintexts to successfully obtain AES secret key parts under the simplified scenario conditions.

Regarding a full key recovery, that was possible in simplified scenarios but seemed very difficult for now to be done with the proposed approach in a realistic scenario. These difficulties come from two main factors. Firstly there is the trouble of detecting a small leakage, as a result of randomising a big portion of the plaintext. This is a crucial step to recover the key, as it is how the bytes are isolated in order to attack only a certain key part, and how the energy consumption from the consequent rounds

is prevented from hiding the first round's leakage. While the theory suggests that it can be done, better acquisitions would be required to do so. Secondly, there is the classification performance, that although it proves capable of distinguishing some templates, it does not allow a correct matching of all the 9 templates with a limited or feasible number of power traces and plaintext pairs.

While proving unsuccessful for a real threat as the attack is at this phase, it has happened before in many areas including information security that consequent improvements to an idea led it from a prototype to a real and applicable case. This work also suggests how those improvements can be measured, by increasing the difficulty of the simplified scenario in different ways.

Finally, one last conclusion supported by this research is that future upgrades to RAPL and similar software-based energy measurement tools must have information security taken into account. For example, refinements in the resolution or sampling rate to this ADCs can be enough to improve the quality of the acquisitions in order to make a full key recovery possible.

As a result of the limited scientific literature related to this subject, multiple times it happened that a branch of ideas happened and it was required to choose a path to stick to in order to proceed with the research. Given the time and the objectives of this thesis it proved impossible to explore all those other ways that could lead to significant improvements to this type of attacks.

First, given the big amount of widely used encryption algorithms, a different one could be analyzed, including asymmetrical encryption. For this thesis, AES was chosen as it is the most used symmetrical one, and power analysis techniques already exist for it. However, there are other options, simpler than AES and still used today, that can be analyzed and possibly prove less resistant against this attack. Similarly, the same applies for the architecture or model used. It was seen in Chapter 6 that a different processor from the same architecture had very different experimental results, and therefore it is also a subject worth investigating.

Under the data acquisition topic, two important ideas were left unexplored. The first one is the study of multiple threads to allow encrypting and measuring at the same time, providing a better simulation of a real-case attack. The other tackles the problem of the existing long-term energy consumption shift independent of the processed data. While doing the acquisition campaigns, it was noted that long capturing periods led to a shift of the mean value of the distribution, due to unwanted noise either coming by other processes or from external factors. In the course of this work and as mentioned before, this was surrounded by changing the plaintext every 7 minutes. A different approach left unexplored consists of using a periodic specific operation to measure the background noise at a certain interval, and using it to adjust the captured values and it's resultant distribution.

In terms of post processing the acquired data, a more profound study towards grouping and filtering the samples is proposed. Regarding the function that matches the templates to the power traces, different improvements can also be done. To start, a way to contour the difficulties related to the increase of the number of Hamming weights is proposed, as the miss-classifications have a tendency to hit templates adjacent to the correct one. Secondly, being basically a classification problem, it would be a great contribution to see the performance of artificial intelligence and machine learning algorithms for such a task. These algorithms have been proving more and more efficient, gaining popularity in information

security among its many other applications. In fact, during the last few years, these algorithms have inspired new side-channel attacks.

Bibliography

[1] P. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. *Advances in Cryptology - CRYPTO96*, pages 104–113, 1996.

[2] P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. *Advances in Cryptology - CRYPTO99*, page 388–397, 1999.

[3] S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer, 01 2007. ISBN 978-0-387-30857-9. doi: 10.1007/978-0-387-38162-6.

[4] S. B. Örs, E. Oswald, and B. Preneel. Power-analysis attacks on an fpga - first experimental results. In C. D. Walter, Ç. K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2003*, pages 35–50. Springer Berlin Heidelberg, 2003.

[5] L. Yan, Y. Guo, X. Chen, and H. Mei. A study on power side channels on mobile devices. pages 30–38, 12 2015. doi: 10.1145/2875913.2875934.

[6] J. Bonneau and I. Mironov. Cache-collision timing attacks against aes. In L. goubin and M. Matsui, editors, *Cryptographic Hardware and Embedded Systems - CHES 2006*, pages 201–215. Springer Berlin Heidelberg, 2006.

[7] Z. He and R. Lee. How secure is your cache against side-channel attacks? pages 341–353, 10 2017. ISBN 978-1-4503-4952-9. doi: 10.1145/3123939.3124546.

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

[9] *Second Generation Intel Xeon Scalable Processors, Volume one: Eletrical*. Intel, 2019.

[10] S. Voinigescu. *High-Frequency Integrated Circuits*. Cambridge University Press, 2013.

[11] V. M. Weaver. Measuring energy and power with papi. *41st International Conference on Parallel Processing Workshops*, page 262–268, 2012.

[12] H. Mantel, J. Shickel, A. Weber, and F. Weber. Vulnerabilities introduced by features for software-based energy measurement. Technical report, Department of Computer Science, Technische Universitatät Darmstadt, Germany, 2017.

[13] *Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide*. Intel, 2009.

[14] D. Terpstra, H. Jagode, H. You, and J. Dongarra. Collecting performance data with papi-c. In A. S. Matthias S. Muller, Michael M. Resch and W. E. Nagel, editors, *Tools for High Performance Computing*, pages 157–173. Springer Berlin Heidelberg, 2010.

[15] E. Oswald, L. Mather, and C. Whitnall. Choosing distinguishers for differential power analysis attacks.

[16] H. Maghrebi, O. Rioul, S. Guilley, and J.-L. Danger. Comparison between side-channel analysis distinguishers. In T. W. Chim and T. H. Yuen, editors, *Information and Communications Security*, pages 331–340. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-34129-8.

[17] B. Gierlichs, L. Batina, P. Tuyls, and B. Preneel. Mutual information analysis. In E. Oswald and P. Rohatgi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2008*, pages 426–442, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-85053-3.

[18] H. Zhao, Y. Zhou, F.-X. Standaert, and H. Zhang. Systematic construction and comprehensive evaluation of kolmogorov-smirnov test based side-channel distinguishers. In R. H. Deng and T. Feng, editors, *Information Security Practice and Experience*, pages 336–352, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-38033-4.

[19] H. Farouk and A. F. Abdelmoneim. Differential power analysis attack using the upper tailed t-test distinguisher. pages 150 – 155, 08 2010. doi: 10.1109/SIBIRCON.2010.5555329.

[20] H. Gamaarachch and H. Ganegoda. Power analysis based side channel attack. Technical report, Department of Computer Engineering, Faculty of Engineering, University of Peradeniya, 2018.

[21] S. Chari, J. R. Rao, and P. Rohatgi. Template attacks. In B. S. Kaliski, ç. K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002*, pages 13–28, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. ISBN 978-3-540-36400-9.

[22] B. Gierlichs, K. Lemke-Rust, and C. Paar. Templates vs. stochastic methods. In L. Goubin and M. Matsui, editors, *Cryptographic Hardware and Embedded Systems - CHES 2006*, pages 15–29, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-46561-4.

[23] M. Elaabid, S. Guilley, and P. Hoogvorst. Template attacks with a power model. *IACR Cryptology ePrint Archive*, 2007:443, 01 2007.

[24] T. Chong and K. Kaffes. Hacking aes-128. Stanford University.

[25] S. Belaïd. *Security of cryptosystems against power-analysis attack*. PhD thesis, Cryptography and Security [cs.CR]. Ecole normale supérieure - ENS PARIS, 2015.

[26] T. de Cnudde. *Cryptography Secured Against Side-Channel Attacks*. PhD thesis, KU Leuven – Faculty of Engineering Science, 2018.

[27] A. Thillard. *Countermeasures to side-channel attacks and secure multi-party computation*. PhD thesis, Cryptography and Security [cs.CR]. PSL Research University, 2016.

[28] T. Popp and S. Mangard. Masked dual-rail pre-charge logic: Dpa-resistance without routing constraints. In J. R. Rao and B. Sunar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2005*, pages 172–184. Springer, 2005.

[29] A. A. Ding, L. Zhang, Y. Fei, and P. Luo. A statistical model for higher order dpa on masked devices. In L. Batina and M. Robshaw, editors, *Cryptographic Hardware and Embedded Systems - CHES 2014*, pages 147–169. Springer Berlin Heidelberg, 2014. ISBN 978-3-662-44709-3.

[30] T. B. Paiva, J. Navaridas, and R. Terada. Robust covert channels based on dram power cnsumption. In Z. Lin, C. Papamanthou, and M. Polychronakis, editors, *Information Security*, pages 319–338. Springer International Publishing, 2019.

[31] C. O'Flynn and A. Dewar. On-device power analysis across hardware security domains: Stop hitting yourself. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019:126–153, Aug. 2019. doi: 10.13154/tches.v2019.i4.126-153. URL `https://tches.iacr.org/index.php/TCHES/article/view/8347`.

[32] K. Khan, M. Hirki, T. Niemi, J. Nurminen, and Z. Ou. Rapl in action: Experiences in using rapl for power measurements. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, 3, 01 2018. doi: 10.1145/3177754.

[33] T. Schneider and A. Moradi. Leakage assessment methodology. In T. Güneysu and H. Handschuh, editors, *Cryptographic Hardware and Embedded Systems – CHES 2015*, pages 495–513, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg. ISBN 978-3-662-48324-4.

[34] D. Chicco and G. Jurman. The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation. *BMC Genomics*, pages 6–18, 2020.