POLITECNICO DI TORINO

Department of Control and Computer Engineering (DAUIN)

Master's Degree Thesis

Empirical Evaluation of the Resilience of Novel S-Box Implementations Against Power Side-Channel Attacks

Supervisor
Paolo Ernesto Prinetto

Candidate
Samuele Yves Cerini

Academic Year 2020/2021

Acknowledgements

I would like to thank Professor Prinetto for the opportunity to work on this thesis, whose topic has interested and inspired me from the beginning to the end of this journey. I would also like to thank the whole team, especially Gianluca and Nicolò who have continuously assisted me. Your help, your continuous dedication, our laughter and the family environment you have built have been of great support during these 6 months. Thank you!

I would like to thank my friends and my girlfriend, for all these amazing years we have spent together and for all the years we will spend in the future.

Finally, I would like to thank my family, from my parents to my uncles, for the continuous support they have always given me, for everything they have taught me and for the fantastic life they have made it possible for me to live so far.

To all of you, my deepest and most sincere gratitude

Abstract

The increasing pervasiveness of embedded and IoT devices and the consequent growth in the amount of data to be processed and exchanged mandates a systematic use of cryptography to ensure confidentiality, authenticity and integrity of communications. Over time, cryptographic techniques have improved considerably, so that cryptanalytic attacks have been made not feasible in reasonable times. On the other hand, this pervasive diffusion of devices has put the attacker in the position of being more likely in *physical proximity*, or in some cases, in *possession* of the components. This has enabled for a plethora of attacks aimed more at the weaknesses in the physical implementations of devices, rather than those of mathematical algorithms, in a way that simply could have not been accounted by cryptographers.

Researchers have demonstrated that secrets processed inside chips can be retrieved with a novel class of attacks, called *side-channel analysis*. These are carried out by observing physical parameters emitted by a hardware component such as the time taken for a cryptographic operation, the power consumption required, or the electromagnetic and acoustic emissions. By combining such physical "clues", an attacker can possibly retrieve the secret without recurring to cryptanalytic exploits or brute-force attacks.

Since such physical quantities cannot be completely masked to the surrounding environment, obfuscation approaches can make attack attempts harder enough to make them not convenient. Along with existing physical-layer countermeasures adopted by chip designers, cryptographers devised new variants of the best-known cryptographic algorithms, such as `AES`, trying to reduce side-channel emissions from the logical level.

Specifically for `AES`, the main target for improvements is the `S-Box` structure, the core non-linear component of the cipher, which already provides the necessary entropy against cryptanalysis (following Shannon's *Confusion and Diffusion* principle). Recent studies combine the existing properties of `S-Box`es with the resilience against side-channel attacks, demonstrating the effectiveness of such novel structures from a mathematical point of view. Still, an in-field empirical proof of such enhancements is not clearly present in the state of the art.

This work aims to explore these latest improvements to the `AES` algorithm, studying novel `S-Box` implementations and finally empirically demonstrating their theoretical claims and effectiveness against power side-channel attacks. To fulfill this aim, a dedicated platform has been employed: the *ChipWhisperer™ Lite* board. *ChipWhisperer™* has been designed to perform side-channel attacks with respect to the device power consumption and it is widely used by both researchers and actual hardware hackers due to its extreme flexibility, overall capabilities and extremely low-cost properties.

The experiments were conducted on a pre-existing software implementation of the AES-128 algorithm, where the original S-Box was replaced with the new structures. The new variants of the algorithm were then run on an Atmel 8-bit microcontroller, using the *ChipWhisperer™* board to collect the power traces related to the various executions. The obtained results were finally compared to those produced by the standard version of `AES`.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

The widely known *Spectre* [8] and *Meltdown* [9] attacks demonstrated that side-channel analysis can be exploited, in conjunction with other techniques, to target consumer-grade Personal Computers, up to large scale mainframes. Recent Proof of Concepts [10] demonstrated the feasibility of these attacks using Web technologies like JavaScript, proving the pervasiveness this class of attacks can leverage in real world scenarios. The hardware layer of any modern device plays a primary role in Information System security: it represents, by construction, *the last line of defense* against intrusions [11]. Hardware is directly or indirectly the base all the other layers rely on. Therefore, a hardware-targeting attack may render useless all the defences implemented in the upper layers (like the ones related to *system* and *application software*) [12].

Given the nature of these attacks, it stands to reason that having the broadest possible knowledge of side-channel analysis, including possible countermeasures, is of extreme importance for digital designers, application engineers, and hardware security experts. Indeed, one the main authors in the field of side-channel analysis, Kocher, insists on the importance of having a systematic approach to security at each implementation level: the intercommunication between the various development groups (cipher designers, hardware designer and software engineers) is essential as security faults often involve unanticipated interactions between components designed by different people [13].

Over the last decade, academic research confirmed several times the effectiveness and the consequent dangerousness of attacks targeting cryptographic implementations on physical devices. Nowadays, side-channel attacks are considered to be part of cryptanalysis techniques in the same way as linear and differential analysis. Moreover, the convenience that characterizes this type of attacks poses significant risks in the use of IoT devices and embedded systems, especially given the pervasiveness that these have in everyday life. For instance, while linear or differential cryptanalysis needs terabytes of plaintext-ciphertext pairs to break a block cipher, differential power analysis can leverage as just as 2000 bytes of such pairs to break the related physical implementation [14]. In general, for a malicious actor, the choice of which attack to use is often dictated by how practical the attack is: in the case of an embedded device this unquestionably falls on side-channel attacks.

In recent years, several countermeasures against this class of attacks have been proposed. However, the performance impact that these solutions have on real embedded devices has been proved to be significant and difficult to circumvent. The need for lighter countermeasures has shifted the interest from the implementation/physical level to the logical and mathematical one, the same in which encryption algorithms are specified. The main motivation behind this choice considers resistance to side-channel attacks in the same way as resistance to classical cryptanalysis, treating it as a key parameter to be taken into account for the successful development of old and new block cipher algorithms.

Many of the recent proposals are directly targeting S-Box designs. Since these structures already provide many properties against classical cryptanalysis, it is therefore reasonable to integrate in their design possible countermeasures against side-channel attacks. Various publications observed that metrics representing the capability of an `S-Box` structure to prevent classical cryptanalysis attacks are in opposition with theoretical metrics used to denote the resistance against side-channel analysis.

These observations carried to a slow abandonment of cryptographic design methodologies like algebraic ones, which are able to produce `S-Box` structures with excellent properties against cryptanalysis but unable to achieve trade offs towards side-channel resistance properties. Therefore, new methodologies with the goal of building novel `S-Box` structures arose, like the ones based on chaotic maps and heuristic methods. From this point on a myriad of new `S-Box` proposals were made, claiming of having found a good trade-off among these metrics.

However, we observed that many of these proposal never left the theoretical environment and computer simulations, de facto rarely proving their theoretical claims with everyday embedded implementations. The work carried out in this thesis therefore consists in empirically verifying the resilience to side-channel attacks of some of the latest proposals of `S-Box` structures. The objective is to test the structures examined in the context of a real attack scenario carried out by means of specialized but low cost and affordable equipment, like the *ChipWhisperer*™ board.

This thesis is organized in five different chapters.

Chapter 2 provides an overview on the `AES` algorithm and classical cryptanalysis methodologies. Chapter 3 gives the necessary basis on side-channel analysis, targeting three of the most powerful and known power attack methodologies. Chapter 4 provides an overview on the latest cryptographic proposals targeting `S-Box` structures, defining the criteria and methodologies used to harden their designs against side-channel attacks. Chapter 5 and 6 finally present the analysis conducted for this thesis work, providing interesting observations on the results obtained and suggesting possible future improvements.

Chapter 2

Advanced Encryption Standard (AES)

2.1 Introduction

The Advanced Encryption Standard (AES) is a specification for data encryption drafted by the NIST (National Institute of Standards and Technology) which finds its actual implementation in the Rijndael algorithm. Rijndael is an iterated block cipher[1] designed by Vincent Rijmen and Joan Daemen, two belgian cryptographers that issued it to an open selection process proposed in 1997 and finally concluded in the year 2000. This process had the goal to find an algorithm that could supersede the standard in use at that time: `DES` (Data Encryption Standard).

`DES` was initially designed by Horst Feistel (to which we owe the concept of *Feistel's Networks*) and then enhanced by the NSA. After being standardized in 1977, it was no longer considered secure starting from the late '90s following repeated brute-force demonstrations of its cryptographic weakness. It was in that years (January 1997) that the National Institute of Standards and Technology (NIST) initiated the search for a replacement for `DES` [15]. The requirements for the new standard were the following:

- a 128-bit block cipher with three possible key sizes: 128, 192 and 256 bits;
- a public and flexible design, in contrast to the `DES` algorithm, which captured the suspects of the cryptographic community due to the presence of classified elements and the alleged presence of backdoors inserted by the NSA;
- at least as secure as two-key triple-`DES`, which was the latest `DES` iteration in use at that time to be considered still secure;
- available royalty-free worldwide;

Rijndael, competing against four other finalist algorithms (MARS, RC6, Serpent, and Twofish), was eventually chosen as the winner of the selection process and became a U.S. federal standard in 2001.

In order to meet the requirements proposed by NIST, some additional features of Rijndael were discarded. For example, the initial specification drafted by Rijmen and Deamen called for different block sizes of 128, 192 and 256 bits: the latter two block sizes were discarded in favor of a constant block size of 128 bits (consequently, the difference on the variants of `AES` simply relates on the size of the encryption key, not the block size).

[1] A block cipher is a deterministic algorithm that operates on a fixed-length block of information and produces, using the cipher key, a different block of the same size.

2.2 The Algorithm

AES is based on a *Substitution-Permutation Network* (*SP-Network*), requiring 10, 12 or 14 encryption rounds, depending on the length of the key (128, 192 and 256 bits, respectively). This number of rounds has been determined by looking at the maximum number of rounds for which shortcut attacks have been found [2], finally adding a considerable security margin [16]. For instance, in the case of AES-128, 4 of the total 10 rounds have been added as a security margin [3].

All the various steps of the algorithm operate on an internal data structure, called *State*. The *Initial State* corresponds to the plaintext (ciphertext) to be encrypted (decrypted), the *Final State* corresponds to the encrypted (decrypted) message and the *Intermediate State* refers to the intermediate cipher result. In all variants of AES, both the State and the cipher key can be visually represented as a matrix of Bytes with 4 rows and 4 columns [4].

Differently from DES, each AES round does not follow a *Feistel's Structure*, but is composed of three distinct invertible uniform transformations, called *layers* [16] [17]:

1. **Linear Mixing Layer**: guarantees high *diffusion* over multiple rounds and consists of two sublayers, both of which perform linear operations:
   (a) **ShiftRows Layer**: permutes the Bytes of each row of the State matrix;
   (b) **MixColumn Layer**: combines the columns of the State matrix, mixing them as arrays of 4 Bytes.
2. **Non-Linear Layer**: each Byte of the State is nonlinearly transformed using lookup tables (S-Boxes) with special mathematical properties. This introduces *confusion* in the data processed;
3. **Key Addition Layer**: XOR operation between the *Round Key*[5] and the *State*.

In general, as for all block ciphers based on *SP-Networks*, the substitution layer(s) operate by mapping small chunks of data, introducing local *confusion* into the cipher thanks to their highly nonlinear properties. On the other hand, the permutation layer(s) operate on the entire block with simple linear transformations, *diffusing* the effects of the mappings enforced by the substitution layer(s) [18].

2.2.1 Algorithm Steps - Encryption

Each round of the algorithm is composed of 4 main operations: SubBytes, ShiftRows, MixColumns and AddRoundKey. However, the rounds are not repeated equally 10, 12 or 14 times: each variant of AES firstly performs a AddRoundKey operation (also known as *Key Whitening*) and only then it executes *R-1* rounds (where *R* is the total number of rounds). Finally, the last round completes 3 operations out of the 4 available, skipping the MixColumns one. The round distribution of an entire execution of the AES algorithm is summarized in Figure 2.5.

[2] A shortcut attack is an attack more efficient than exhaustive key search. [16]

[3] For the AES variants with 192 and 256 bits keys the number of rounds is increased to prevent shortcut attacks, which, conversely to exhaustive attacks, can afford to be less efficient in case of longer keys [16]

[4] The original specification of Rijndael specified a variable number of columns, depending on the block size. As mentioned in 2.1,the standardization process that lead to AES imposed a block length of 128 bit, therefore fixing the number of columns of the State to 4

[5] A Round Key is an intermediate key derived by the *Key Schedule* algorithm from the main cipher key. See 2.2.2

`SubBytes`

This operation performs a substitution of every Byte present in the State matrix, following a precise Substitution Table (called `S-Box`) that acts as a LUT (Look-Up Table). The `S-Box` is the only nonlinear element of `AES`: it implements a bijective mapping, meaning that each of the $2^8 = 256$ possible input values is one-to-one mapped to one output Byte. This last property implies the possibility to uniquely reverse the S-Box, therefore allowing decryption. `AES` `S-Box`es have a strong algebraic structure, based on two main operations: a Galois field inversion (in the field $GF(2^8)$) and an affine mapping [17]. Moreover, unlike `DES`, where 8 different `S-Box`es are applied, `AES` leverages a unique `S-Box` for the entire cryptographic operation. The resulting lower memory footprint allows for much lighter software implementations, making encryption possible even for small embedded systems and IoT devices. The `SubBytes` operation is pictured in Figure 2.1.

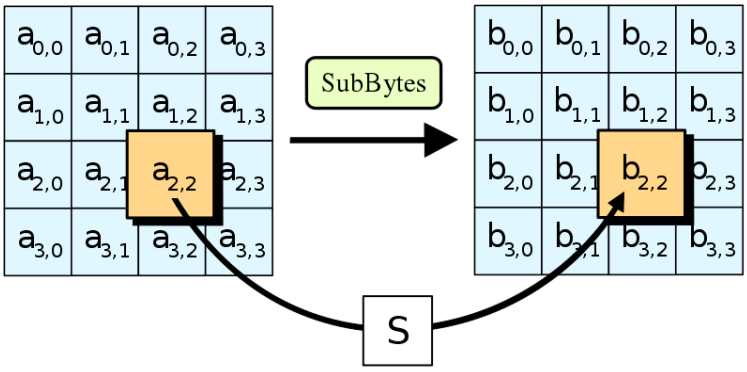

Figure 2.1: The `SubBytes` operation [1].

`ShiftRows`

This operations provides the *diffusion* properties of `AES` by rotating the Bytes over each row of the State matrix. The first row is not rotated, the second one rotates over 1 Byte, the third one rotates over 2 Bytes and the third one rotates over 3 Bytes (see Figure 2.2).

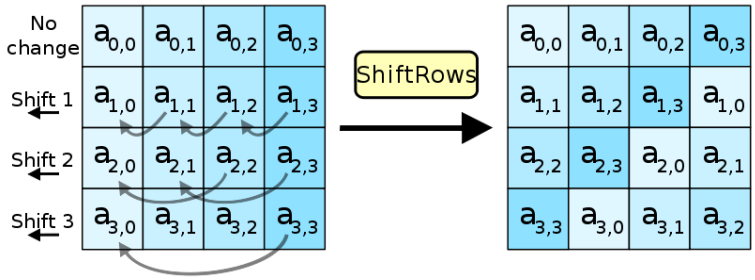

Figure 2.2: The `ShiftRows` operation [2].

`MixColumns`

Like the `ShiftRows` step, the `MixColumns` operation provide the necessary *diffusion* to the algorithm. In this case, every column of the State matrix is treated as a vector and multiplied by a fixed 4×4 matrix containing constant entries. The result is a linear

transformation of the State. The multiplication and addition of the coefficients is done in $GF(2^8)$: both operations can be implemented quite efficiently, as the addition consists of a `XOR` operation and the multiplication can be realized using bitwise shifting and modular reduction, or by simply leveraging additional memory to store precomputed LUTs (see Figure 2.3).

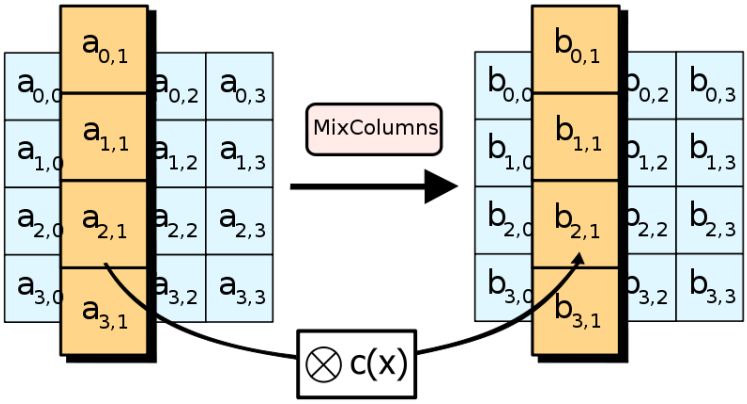

Figure 2.3: The `MixColumns` operation [3].

`AddRoudKeys`

In this step every single Byte of the State is `XOR`ed with the corresponding Bytes that compose the Round Key (see Figure 2.4).

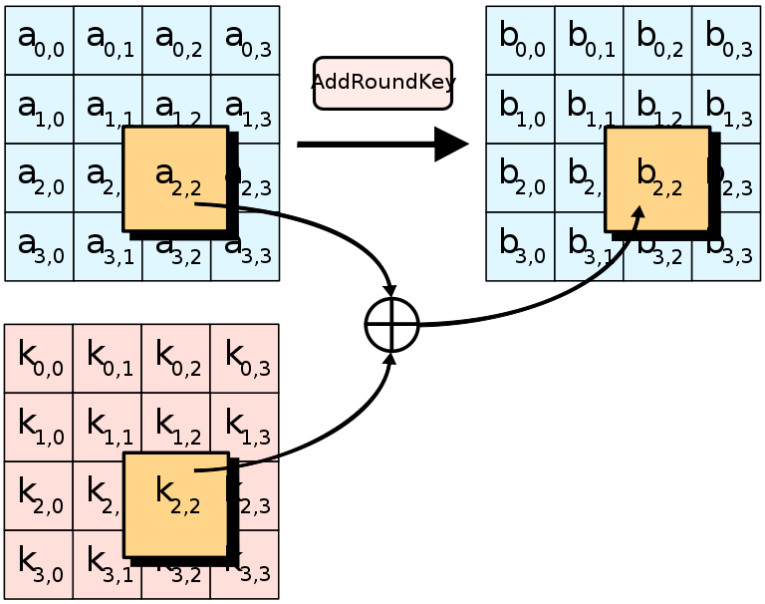

Figure 2.4: The `AddRoundKeys` operation [4].

2.2.2 Key Schedule Operation

The *Key Schedule* is an internal algorithm used by `AES` to derive multiple *Round Keys* from the original input key of the cipher, to be used in the `AddRoundKeys` steps. The number of round keys to be derived is equal to the number of rounds plus one (since the

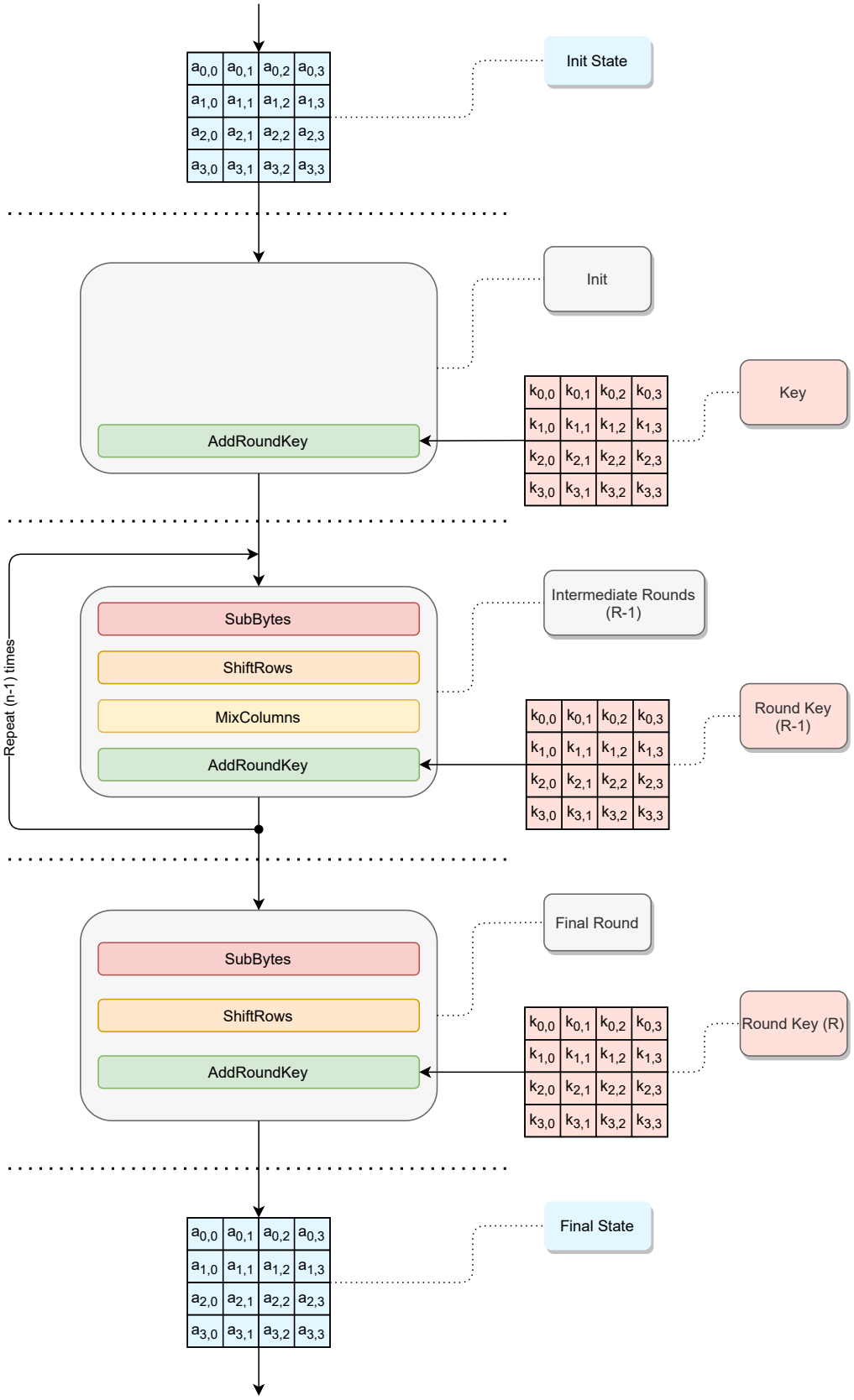

Figure 2.5: Round organization in a complete execution of an `AES` encryption.

initial *Key Whitening* operation must be considered, so as the $R-1$ "complete" rounds and the final round). As an example, for a key length of 128 bits, the total number of rounds is $R = 10$, giving 11 round keys, each of 128 bits [17].

The algorithm slightly changes depending on the `AES` variant in use. For instance, `AES-128` has the first 128-bit round key which is exactly equal to the main cipher key: this is not true for the other two `AES` variants (the first round key of `AES-196` is equal to the main key only for the first 128 bits).[6]

Two different implementations exists for the *Key Schedule* algorithm:

1. **Precomputation**: All round keys are computed before the actual encryption (decryption) operation and stored in memory, each to be retrieved later when needed. This approach is often taken in PC and server implementations of `AES`, where large pieces of data are encrypted under one key. Such an implementation is not desirable on a device with limited memory resources, such as a smart card;

2. **On-The-Fly**: Every round key is derived only when required during the encryption (decryption) process. This reduces the memory footprint of the key schedule algorithm but introduces computational overhead.

2.2.3 Algorithm Steps - Decryption

Since `AES` is not based on a *Feistel's Network*, the decryption operation cannot be performed by simply re-run the same algorithm: an inverse implementation of every round is needed. The structure of Rijndael is such that the sequence of transformations of its inverse is equal to that of the cipher itself, with the transformations replaced by their inverses and a change in the *Key Schedule* [16]. Therefore, the 4 main steps that compose each round, once inverted, are now called `InvSubBytes`, `InvShiftRows`, `InvMixColumns` and `AddRoundKey`. This last transformation doesn't need further modifications as the internal `XOR` operation is intrinsically invertible by reapplying the function itself.

The inverse structure of a generic round is composed as such:

1. `AddRoundKey`;

2. `InvMixColumns`;

3. `InvShiftRows`;

4. `InvSubBytes`.

Since the last encryption round does not perform the `MixColumns` operation, the first decryption round also does not contain the corresponding inverse layer (`InvMixColumns`). Finally, an additional `AddRoundKey` step is performed as last operation for the decrypting algorithm, as opposed to the encrypting algorithm, where this operation (*Key Whitening*) is performed first.

2.3 Cryptanalysis

Cryptanalysis is a set of techniques used:

- *to test* the robustness of an encryption scheme and its secret key by attacking it with various attack models;

[6] This peculiarity of `AES-128` is one the main reasons why this variant has been adopted for the purposes of this thesis. More details will be given in the following chapters.

- *to break* an encryption scheme revealing the plaintext and/or the secret used to encrypt it.

It is therefore an essential stage in the development of secure algorithms: in order to design a strong cipher, the cryptographer has to understand where the potential weaknesses are [18].

In general, cryptanalysis leverages mathematical techniques to analyse encryption schemes from their logical specification. Additionally, cryptanalysis also includes side-channel attacks in order to target the weaknesses at the implementation level (see Chap. 3). From a point of view of the bare logic specification, a cipher should be secure even if the attacker knows every detail of the algorithm, following *Kerckhoffs' Principle*. This principle actually discourages the use of the *security by obscurity* technique, which has been demonstrated to be a *weak* design principle: an algorithm which security relies only on the obfuscation of its specification can be completely broken as soon as it is secret design gets reversed-engineered or leaked through other means [17]. This is why modern cryptographic schemes (like `AES`) are completely open source and available for analysis from the cryptographic community.

2.3.1 Methodology Overview

In the case of block ciphers like `AES`, the goal of cryptanalysis is to break the cipher by recovering the plaintext partially or in its entirety. An even ambitious goal would be to retrieve the cipher key: in this scenario an attacker can leverage the retrieved secret to encrypt/decrypt all the messages from the moment the key was revealed onward, harming the confidentiality and authenticity of the user's communications.

Cryptanalysis can be carried on following various attack models [18] [7], including:

- **Ciphertext-only attack**: The attacker can only observe ciphertexts produced as output by the encryption algorithm. Since this scenario is indistinguishable from the normal behavior of the algorithm, block ciphers succumbing this model of attack are considered to be very weak;
- **Known-plaintext attack**: The attacker has access to (parts of) the plaintext related to the captured ciphertext. On the other hand, the attacker does not have any control of the plaintext to be encrypted;
- **Chosen-plaintext attack**: The attacker has control over the plaintexts generated, which are then feeded to the encrypting algorithm. The resulting ciphertexts are then observed;
- **Chosen-ciphertext attack**: Similarly to the previous model, the attacker can carefully craft a ciphertext, submit it to the block cipher for its decryption and observe the plaintext produced as output;
- **Adaptively chosen-plaintext attack**: The attacker has the possibility to choose and send multiple plaintexts, possibly creating each plaintext according to the results obtained from the observations of the previously obtained ciphertexts;
- **Adaptively chosen-ciphertext attack**: Similarly to the previous model, the attacker submits multiple ciphertexts to the decrypting algorithm, adapting its decisions following the observations retrieved during the entire attack procedure;
- **Related-Key attack**: Similar to a chosen-plaintext attack, in this model the attacker can compare and analyze two different ciphertexts (plaintexts) obtained

using two different secret keys, having only the information about their relationship (for instance, the two keys may differ only by a single bit).

It is worth to point out that an attacker cannot compute the correspondence between a plaintext and a random ciphertext if the key value is unknown. The only way to keep track of which plaintext (or ciphertext) has produced which ciphertext (or plaintext) is to *query* the cipher sequentially, storing the plaintext-ciphertext pairs obtained. A system that replies to these *queries* is called an *oracle* (respectively, an *encryption oracle* and a *decryption oracle*). This *query*-based attack will be the one in use for this thesis work, as it is widely used to carry on side-channel attacks.

2.3.2 Generic Attacks

Given the limited key and block lengths that characterize real (i.e., implementable) block ciphers, *brute-force* and *dictionary* attacks are considered to be the most generic class of attacks a malicious actor can pursue against an encryption scheme. Indeed, such methodologies can break a block cipher simply by treating it as a black box, without considering the actual details of the internal algorithmic implementation.

A *brute-force* attack, also called *exhaustive key search*, simply iterates an encryption (decryption) process on a given plaintext (ciphertext) trying all possible key combinations, comparing each time the obtained ciphertext (plaintext) with the "golden" one (i.e., the one obtained with the secret key). This attack is made ineffective [7] by simply increasing the number of bits of the cipher key: nowadays an 80-bit key is considered to be the bare minimum to prevent *exhaustive search*. For instance, the "weakest" variant of `AES` (`AES-128`) leverages a 128-bit key. All the optimizations for this methodology aim to reduce the number of necessary iterations, reducing the computational workload and possibly leveraging the time-memory tradeoff, as proposed by Hellman in [19].

In a *dictionary* attack, an attacker can submit multiple plaintexts (ciphertexts) to the block cipher obtaining back the related ciphertexts (plaintexts), building a table of plaintext-ciphertext mappings (a dictionary) for that specific key. If a malicious actor is able to submit all the possible plaintexts (the cardinality of the entire plaintext space follows the relation 2^b, whereas b corresponds to the block length in bits) then the consequent dictionary can be used to decrypt any future message encrypted with the same secret key. In reality, it is not even necessary to complete the dictionary by submitting all possible plaintexts: whenever a block cipher outputs the same ciphertext block twice, it leaks information about the plaintexts. This kind of repetitions in a random set of b-bit blocks starts to occur frequently when the number of blocks exceeds $2^{b/2}$ (a consequence of the *birthday paradox* [8]). It is therefore not advisable for any user to encrypt more than $2^{b/2}$ blocks with the same secret key [7]. Similarly to a *brute-force* attack, the *dictionary* attack is made ineffective by simply increasing the number of bits of the block length.

A comparison between these two methodologies can highlight the tradeoffs they provide: with a *brute-force* attack the attacker needs a single memory location, storing the single captured plaintext-ciphertext pair: the memory requirements are thus very low. On the other hand, each iteration (out of the 2^k possible one, being k the key length expressed in bits) requires time to complete and, overall, lots of computational power. Conversely, a *dictionary* attack, assuming the dictionary already built and available for use, requires multiple memory locations (one for each plaintext-ciphertext pair, for a total

[7] i.e., not feasible in reasonable time due to the high computational complexity

[8] In cryptography, the same principle is exploited when evaluating the probability of collisions for a given hash function

memory amount of $2b2^b$ bits) but a negligible time complexity given the single access needed to look-up for one of the two values that compose a pair.

As a remark, both *brute-force* and *dictionary* attacks are theoretically impossible to prevent due to the intrinsic properties of block cipher algorithms. However, nowadays ciphers are built with a large number of key and block bits, making this class of attacks unfeasible with the current computational resources.

2.3.3 Shortcut Attacks

As opposed to *Generic Attacks*, *Shortcut Attacks* leverage the understandings of the internal algorithm implementation to reduce the computational complexity. For a block cipher, the feasibility of a shortcut attack is usually considered to be a significant vulnerability. Since both brute-force and dictionary attacks set an upper bound on both time and memory requirements, an attack that has an overall reduced complexity is considered to be a shortcut attack.

In general, this class of attacks is conducted in two main steps. The first step tries to find and leverage properties that allows the attacker to overcome the ability of the block cipher to make every ciphertext produced completely random. If such a property is found to be also reproducible, the attacker can now formalize a methodology that generalize an attack for a reduced variant of the encryption scheme, omitting a certain number of rounds for simplicity. The goal of this second phase of the attack is to retrieve, partially or completely, a round key. If found, the attacker can continue with the same procedure on another round of the algorithm or can now leverage an exhaustive search, supported by the lower computational complexity provided by reduced exploration space.

Differential Cryptanalysis

Differential Cryptanalysis was firstly introduced by Biham and Shamir to attack `DES` in 1992 [20]. It is a chosen-plaintext attack, meaning that the attacker is able to submit specially crafted plaintexts and to examine the ciphertexts produced as outputs in an attempt to derive the key.

Attack rationale This method focuses on finding datapaths in the block cipher structure that are more likely to occur with certain selected sets of plaintexts. The attack is called "differential" because the plaintexts are selected in pairs in order to satisfy a certain differential relation among their values.

For instance, suppose that x and x' are two messages processed by the same function F: the results obtained are, respectively, $F(x)$ and $F(x')$. Knowing that the difference between two generic input values x and x' is called Δx and corresponds to the `XOR` operation between the two [9],

$$\Delta x = x \oplus x' \tag{2.1}$$

It is possible to compute the difference between the two results obtained with the two computations:

$$\Delta y = F(x) \oplus F(x') \tag{2.2}$$

The fact that an input difference Δx is reflected as an output difference Δy after a transformation F is said to be a *propagation* and the $[\Delta x, \Delta y]$ pair is called *differential* [21]. Differential cryptanalysis bases its rationale on the observations of these pairs, relying on the fact that a given propagation pattern only occurs for certain values of inputs. The

[9] The difference can also be expressed in other ways: the `XOR` difference is however the most common one.

reason is that, in many computations widely adopted in current block cipher algorithms, *deterministic* differential propagations can be constructed even without knowing the key value [7]. In an ideally randomizing cipher, the probability of a differential propagation to occur is $1/2^z$, where z is the number of bits of the input value x. The attack seeks to exploit a scenario where a particular propagation occurs with a very high probability (much greater than $1/2^n$) [21].

To easily describe the main idea behind the attack, let us imagine a scenario involving the `AES AddRoundKey` operation (but the example can be adapted to many other block ciphers). Let us also assume the attacker has a *single* plaintext P available and does not know the value of the key K used. By feeding P to the `AddRoundKey` operation, the plaintext itself is `XOR`ed with the key K, obtaining as a result the State S:

$$S = P \oplus K \tag{2.3}$$

Since the key K is unknown, the result S is also unknown: every information that was in the attacker possession is now lost.

Let us now assume a similar scenario as the previous one, in which the attacker is now in possession of *two* different plaintexts P and P'. By issuing them sequentially to the `AddRoundKey` operation, the attacker obtains:

$$S = P \oplus K \tag{2.4}$$

and

$$S' = P' \oplus K \tag{2.5}$$

Now, the attacker can compute the difference between the two plaintexts P and P' (both known) and the two results S and S' (unknown), following the definitions given in 2.2 and 2.1:

$$\Delta P = P \oplus P' \tag{2.6}$$

$$\Delta S = S \oplus S' = (P \oplus K) \oplus (P' \oplus K) = (P \oplus P') \oplus (K \oplus K) = P \oplus P' = \Delta P \tag{2.7}$$

The above equation demonstrates that:

- the output difference between S and S' can be computed even without knowing the values of S and S' and is equal to the input difference ΔP, because...
- the difference between the two plaintexts is propagated to the outputs through the `AddRoundKey` transformation;
- the secret key is completely removed: the attacker can observe the effects of the encryption operation without taking into account the actual value of the key;

This is possible thanks to the linear nature of the `AddRoundKey` step, as it simply consists of a `XOR` operation.

Since a linear function L satisfies the following property:

$$L(a) \oplus L(b) = L(a \oplus b) \tag{2.8}$$

the difference of two values a and b after a linear computation L can be computed by taking their difference as input to L. This specific case is known as a *Deterministic Differential Propagation in Linear Computation.*

As a consequence of 2.8, given any linear operation, a constant key K and a pair of plaintexts P and P', the input difference Δx is propagated to the output difference Δy with a probability of 1 (i.e., deterministically). It is also true, given the same conditions, that obtaining $\Delta y \neq \Delta x$ is impossible (i.e., the probability is 0). This is true also for a composition of multiple linear operations, since the result of the composition itself is also linear [10].

In general, not every operation of a block cipher is linear (for instance, the `AES S-Box`): in this case, no efficient method is known to derive a high probability differential propagation. The logical consequence for an attacker is to compute the probability of a given $[\Delta x, \Delta y]$ pair by feeding all the possible input values to the nonlinear operation under attack and to list all the resulting output values. This operations aims at building a *Differential Distribution Table* or *DDT*. Each row of the table refers to a specific input difference, while each column refers to a specific output difference. Each entry of the table represents how many of the possible values the input x can assume satisfy the relation:

$$F(x) \oplus F(x \oplus \Delta x_{\text{expected}}) = \Delta y_{\text{expected}} \tag{2.9}$$

The probability of the differential propagation is obtained by dividing the number in each entry with the number of possible values x can assume (given by 2^z).
In general, this probability can also be defined as follows [7]:

$$P[\Delta x \xrightarrow{F} \Delta y] \triangleq \frac{\#\{x | F(x) \oplus F(x \oplus \Delta x) = \Delta y\}}{2^z} \tag{2.10}$$

where:

- F is a function that is either known or unknown to the attacker;
- z is the bit-length of the input value to the function F.

To better illustrate the scenario with a nonlinear operation see example 2.3.3, firstly given in [7] and summarized in Fig. 2.6.

The above definition can be exploited only if the computation size is small, that is, the value of z is small. Namely, 2.10 can be applied for small 4-bit to 4-bit mappings (like example 2.3.3) and even to the `AES S-Box` operation, but it cannot be evaluated for a 128-bit to 128-bit mapping of the entire `AES` cipher. When evaluating a multi-round cipher, the attacker is forced to apply the analysis one round at a time, assuming that the probability of the differential propagation can be evaluated independently for each round: evaluating the *conditional probability*, especially for many rounds, is computationally infeasible.

In a broader view, singularly evaluating the linear and nonlinear operations that make up the structure of the block cipher allows the attacker to find a global *differential* $[\Delta x, \Delta y]$ that propagates from the inputs of the cipher to the inputs of its last round. To build this differential it is necessary to examine highly likely *differential characteristics*: sequences of input and output differences to the rounds so that the output difference from one round corresponds to the input difference for the next round [21]. An highly likely differential characteristic gives the attacker the opportunity to exploit information coming into the last round of the cipher to derive bits from the last layers of subkeys, eventually helping in the procedure of reconstructing the main cipher key (leveraging the knowledge of the key schedule algorithm).

[10] In general, the goal of an attacker exploiting *Differential Cryptanalysis* is to obtain propagations with the highest possible probability (possibly deterministic): this allows to determine the differences between pairs without knowing their respective values, as seen in 2.7. This approach is useful for the analysis because the plaintext values become unknown quickly owing to the key.

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S(x)	C	0	F	A	2	B	9	5	8	3	D	7	1	E	6	4

Table 2.1: TWINE's `S-Box` Mapping in the Hexadecimal Notation

Example: Computing the probability of a given propagation First, it is required to define a cipher operation under attack, keeping in mind that this could be any operation of any block cipher of our interest. In this example the `S-Box` structure defined in the lightweight block cipher TWINE is exploited [22]. The knowledge of the implementation details is not explicitly required: the attacker can observe directly the outputs of the `S-Box` table, without leveraging the knowledge of the applied mapping to infer their values.

Let us assume the goal is to assess the probability of the propagation between the following (given) input difference and output difference:

$$\Delta x = x \oplus x' = \texttt{5} \qquad \Delta y = y \oplus y' = \texttt{S-Box}[x] \oplus \texttt{S-Box}[x'] = \texttt{A} \tag{2.11}$$

$$P[\Delta x \xrightarrow{\texttt{S-Box}} \Delta y] = P[\texttt{5} \xrightarrow{\texttt{S-Box}} \texttt{A}] \tag{2.12}$$

The probability must be computed repeating the procedure for all the possible input values $x \in \{\texttt{0}, \texttt{1}, \ldots, \texttt{F}\}$, enforcing the given input difference Δx and obtaining the second input by simply inverting its relation $x' = \Delta x \oplus x$. The values y and y' are then computed applying the mapping provided by the `S-Box` function:

$$y = \texttt{S-Box}[x], \qquad y' = \texttt{S-Box}[x'] \tag{2.13}$$

It is now possible to compute the output difference 2.2 and check if it matches the expected one ($\Delta y = \texttt{A}$):

$$\Delta y_{\text{obtained}} = \texttt{C} \oplus \texttt{B} \neq \Delta y_{\text{expected}} = \texttt{A} \tag{2.14}$$

By iterating on all the 16 possible values the input x can assume, one can compute the overall probability of the given propagation, which is simply the number of times (out of 16) the input difference $\Delta x = \texttt{5}$ produces the output difference $\Delta y = \texttt{A}$. Given the `S-Box` structure of the TWINE cipher (see Table 2.1), the probability of the propagation of this example is:

$$P[\Delta x \xrightarrow{\texttt{S-Box}} \Delta y] = P[\texttt{5} \xrightarrow{\texttt{S-Box}} \texttt{A}] = 2/16 = 2^{-3} \tag{2.15}$$

This example is summarized in Fig 2.6.

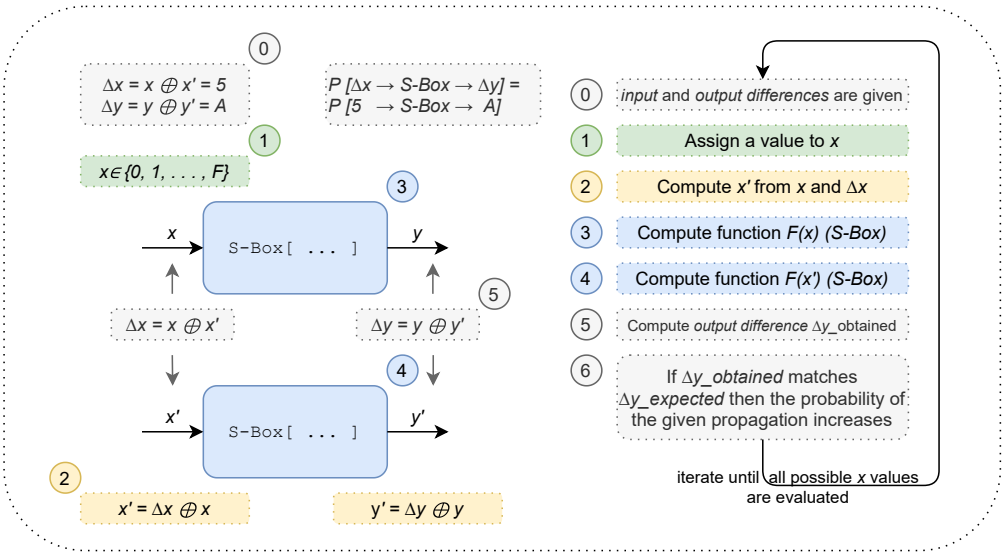

Figure 2.6: Example 2.3.3: computing the probability of a specific differential propagation.

Impossible Differential Cryptanalysis

Following similar principles of the more generic *Differential Cryptanalysis*, this attack exploits differential propagations that are never satisfied.

An impossible differential propagation is defined as follows:

$$P[\Delta x \xrightarrow{F} \Delta y] = 0 \tag{2.16}$$

where Δx and Δy represent the input and output difference, respectively, on both ends of function F. Equation 2.16 indicates that such differential propagation is impossible, since its probability is zero. As an example, an impossible propagation is the one involving an input difference $\Delta x = 0$ (meaning that the two inputs of function F are equal) and an output difference $\Delta y \neq 0$ (meaning that the two outputs of F have different values). A similar example can be obtained considering a bijective function F (such as the `S-Box` of the `AES` block cipher): it is impossible to have a propagation that involves an input difference $\Delta x \neq 0$ and results in an output difference $\Delta y = 0$.

An attack based on an impossible differential characteristic can be built as follows. Let us imagine to have a plaintext pair $[P, P' = P \oplus \Delta x]$ to which corresponds the ciphertext pair $[C, C']$. Let us also assume to have a function F for which the definition of impossible differential propagation holds 2.16. To recover the key, the attacker exhaustively guesses the last round key values and partially decrypts both C and C': the values resulting from this two sequential decryption operations correspond to the outputs values of function F. If the difference between these two values is equal to Δy then the attacker has found a key value that produces the given impossible differential propagation: the key guess is wrong and it can be removed from the set of round keys candidates. Trivially, the attack recovers the right key guess by simply eliminating all the wrong ones.

While the generic variant of differential cryptanalysis tries to find highly probable differential propagations to directly guess the correct value of the round key, the impossible variant, on the contrary, tries to discard all those keys related to impossible differential propagations, finding the correct key guess by exclusion [7].

Linear Cryptanalysis

Introduced in 1993 by Matsui [23] as a theoretical attack on DES (and empirically used shortly after [24]), *Linear Cryptanalysis* is nowadays, together with *Differential Cryptanalysis*, one of the most widely used attacks on block ciphers. Given its importance, it is considered by the cryptographic community as a mandatory step in evaluating the strength of new cipher proposals.

Linear cryptanalysis is a known-plaintext attack, meaning that the attacker has knowledge on a set of plaintexts and the corresponding ciphertexts, without having the chance (or the need) to control which plaintexts (and related ciphertexts) are available. This is an important advantage over differential cryptanalysis as no restrictions are imposed on the plaintexts.

The main rationale behind the attack is to approximate the nonlinear operations of a block cipher with linear functions (like XOR, or any other modulo-2 bitwise operation).

Such a linear function has the following form:

$$P[i_1, i_2, .., i_a] \oplus C[j_1, j_2, .., j_b] = K[k_1, k_2, .., k_c] \tag{2.17}$$

where P represents the input plaintext on a bits, C represents the output ciphertext on b bits and K represents the secret key on k bits. The goal is to find linear approximations like 2.17 that have a high or low probability of occurrence for randomly given plaintexts P and the corresponding ciphertexts C [23]. In fact, in an ideal cipher, any linear equation relating plaintext, ciphertext and key bits would hold with probability $p = 1/2$, proving ideal randomization capabilities. The attack exploits the deviation (or *bias*) from this ideal probability: the further away that a linear expression is from holding with a probability of $1/2$, the better the cryptanalyst is able to apply linear cryptanalysis. If an expression like 2.17 holds with a probability p, the *linear probability bias* can be defined as $\epsilon = p - 1/2$: the higher its magnitude $|\epsilon = p - 1/2|$, the better the applicability of the attack, with a reduced amount of known plaintexts required [21].

Note that both extremes of the probability $p = 1$ and $p = 0$ implies that the related linear expression is a perfect representation of the cipher behavior, hence the cipher has a catastrophic weakness [21].

The attacker concentrates its efforts only on cipher's nonlinear components such as S-Boxes: finding linear relations on cipher's linear operations is trivial. Similarly to what happens with differential cryptanalysis, the procedure to attack a nonlinear operation is to enumerate every possible linear equation relating the S-Box's input and output bits. During this process, a *Linear Approximation Table* is built, allowing the attacker to compute the biases of every linear approximation and to select the best performing ones. The next step of the attack is to build a chain of approximations, called *linear characteristic*. Such a chain can be used to recover the bits of the last round key if it is long enough to cover $R - 1$ rounds (for a R-rounds cipher) and has a suitably large probability bias. To build this *Linear Characteristic*, Matsui proposes the so called *Piling-up Lemma*, which main principle is summarized in the paragraph 2.3.3.

The recover bits of the last round key (also called *target partial subkey* [21]), it is necessary to XOR the corresponding ciphertext bits with all the possible values the target partial subkey can assume. The result of the operation is then back propagated to the S-Box corresponding to the last round we are operating in. This is done for all known plaintext/ciphertext samples and a count is kept considering each value assumed by the target partial subkey. The count is incremented whenever the linear expression holds true for the bits into the last round's S-Box (which are determined by the XOR operation) and the known plaintext bits. Finally, the subkey guess whose count has the greatest absolute

difference from half the number of plaintext-ciphertext pairs represents the most likely set of values for those key bits [21].

Piling-up Lemma [23] [21] Let us consider two random binary variables X_1 and X_2 and their linear and affine relationships: $X_1 \oplus X_2 = 0$ (which leads to $X_1 = X_2$) and $X_1 \oplus X_2 = 1$ (which leads to $X_1 \neq X_2$), respectively. Let us now assume their probability distributions are given by:

$$P(X_1 = i) = \begin{cases} p_1, & i = 0 \\ 1 - p_1, & i = 1 \end{cases} \tag{2.18}$$

and

$$P(X_2 = i) = \begin{cases} p_2, & i = 0 \\ 1 - p_2, & i = 1 \end{cases} \tag{2.19}$$

Assuming the two random variables are independent, then

$$P(X_1 = i, X_2 = j) = \begin{cases} p_1 p_2, & i = 0, j = 0 \\ p_1(1 - p_2), & i = 0, j = 1 \\ (1 - p_1)p_2, & i = 1, j = 0 \\ (1 - p_1)(1 - p_2), & i = 1, j = 1 \end{cases} \tag{2.20}$$

The probability of the linear relation between the two variables is:

$$\begin{aligned} P(X_1 \oplus X_2 = 0) &= P(X_1 = X_2) \\ &= P(X_1 = 0, X_2 = 0) + P(X_1 = 1, X_2 = 1) \\ &= p_1 p_2 + (1 - p_1)(1 - p_2) \end{aligned}$$

Now, let us assume the presence of a *linear probability bias*: it is possible to rewrite both probabilities equations as $p_1 = 1/2 + \epsilon_1$ and $p_2 = 1/2 + \epsilon_2$. Therefore, it follows that:

$$\begin{aligned} P(X_1 \oplus X_2 = 0) &= p_1 p_2 + (1 - p_1)(1 - p_2) \\ &= 1/2 + 2\epsilon_1 \epsilon_2 \\ &= 1/2 + \epsilon_{1,2} \end{aligned}$$

The *Piling-up Lemma* generalizes the above result to multiple random binary variables, from X_1 to X_n, with the respective probabilities going from $p_1 = 1/2 + \epsilon_1$ to $p_n = 1/2 + \epsilon_n$. The lemma also assumes that all n random binary variables are independent [11]:

$$P(X_1 \oplus \cdots \oplus X_n = 0) = 1/2 + 2^{n-1} \prod_{i=1}^{n} \epsilon_i \tag{2.21}$$

or, equivalently,

$$\epsilon_{1,2,\ldots,n} = 2^{n-1} \prod_{i=1}^{n} \epsilon_i \tag{2.22}$$

[11] In reality linear approximations of nonlinear structures like `S-Box`es are not independent between each other and this can have a significant impact on the computation of the linear characteristic probability.

When developing a linear cryptanalysis attack on a cipher, the X_i values actually represent linear approximations of nonlinear operations (like `AES S-Box`). These approximations can be combined together to form new linear expressions and eventually build an entire chain of linear approximations (*linear characteristic*). The *Piling-up Lemma* is therefore crucial to compute the overall composition of the single biases ϵ_i, finally determining the overall bias of the entire chain.

It is interesting to note that if $p_i = 0$ (or 1) for all i, then $P(X_i \oplus \cdots \oplus X_n = 0)$ (or 1). Namely, a linear characteristic composed by n linear approximations perfectly approximate all the block cipher operations if and only if all the single linear expressions that compose the chain perfectly approximate their respective cipher's operations.

Conversely, even a single linear approximation with a probability $p_i = 1/2$ can "destroy" the entire linear characteristic, bringing the entire chain probability to $P(X_1 \oplus \cdots \oplus X_n = 0) = 1/2$, thereby nullifying the bias contributions of all the other approximations.

As demonstrated, the overall bias of a linear characteristic is consequence of all the linear approximations of the `S-Box`es involved (as they are the main nonlinear component of a block cipher's operations). The cryptographic community is constantly trying to improve the resistance against linear cryptanalysis. To do so, the focus is on optimized `S-Box` designs that minimize the biases by keeping their mappings as ideally random as possible. Another technique is to implement round structures that maximize the number of *active* `S-Box`es (so to say, the number of `S-Box`es involved in linear approximations). The design principles of Rijndael are an excellent example of such an approach.

2.3.4 `AES` Security and Countermeasures Against Cryptanalysis

The cryptographic strengths of the `AES` algorithm strongly depends on the choice of the `S-Box` structure. Rijndael's `S-Box` has been designed taking into account the most recent cryptanalysis attacks known at that time, like linear/differential cryptanalysis and algebraic manipulations (for example, interpolations attacks).
The main design criteria took into account by Daemen and Rijmen were the following [16]:

1. Invertibility;
2. Minimisation of the largest non-trivial correlation between linear combinations of input bits and linear combination of output bits;
3. Minimisation of the largest non-trivial value in the EXOR table;
4. Complexity of its algebraic expression in $GF(2^8)$;
5. Simplicity of description.

The final structure allows to obtain a maximum differential propagation probability of 4/256 (or 2^{-6}), even if most entries achieve even lower ratios: either 0/256 or 2/256 [16].

The three main criteria are satisfied by the main mapping $x \Rightarrow x^{-1}$ in $GF(2^8)$, that is however susceptible to algebraic attacks (like interpolation ones) due to its very simple algebraic expression. An additional invertible affine transformation has bee therefore applied on top of it: this allows the `S-Box` to satisfy the fourth criterion too without affecting the first three.

Both Rijndael's cryptographers proposed that the standard `S-Box` they designed could be swapped with similar structures that could satisfy the above criteria, for example, in case of suspicion of a trapdoor present in the block cipher. They both went as far as suggesting that a new `S-Box` structure could not necessarily optimise both linear and differential cryptanalysis criteria (second and third criteria) given the great confidence residing on the overall cipher structure and the conservatively high number of rounds [16].

Chapter 3

Power Analysis

3.1 Introduction to Side-Channel Analysis

As briefly mentioned in the previous chapter, cryptanalysis includes an additional class of attacks whose approach significantly differs from linear, differential and algebraic analysis. This class, called *side-channel analysis*, sets the focus on studying the physical platforms that hosts software or hardware implementations of cryptographic schemes. Such platforms, in the most common scenarios, can be identified as IoT and small embedded devices: in reality, any analog or digital electronic circuit can be attacked. In summary, while classic cryptanalysis exploits the vulnerabilities of the algorithm itself, on a logical level, side-channel analysis studies the vulnerabilities of their implementation, on a physical level.

The main methodology is based on the observation of the physical emissions of an operating device, like electromagnetic, heat and sound emissions. Additional parameters can be observed, although being not properly "emitted", such as the device power consumption or the time taken to execute a specific operation. The observation and retrieval of side-channel information allows the attacker to gather data that, if manipulated with proper methodologies, can eventually lead to the disclosure of the secret key used within the cipher (see Fig. 3.1). This scenario, in which the attacker does not interfere with the observed object, is typical of so-called *passive* attacks. Conversely, *active* attacks expect the attacker to intentionally disturb the computations during the device execution, inserting faults or specially crafted values in the intermediate data treated by the cipher during an encryption/decryption operation (also called *fault attacks*) [7].

Side-channel analysis groups multiple attack implementations, depending on the targeted emission, the most known being *Power Attacks*, *Timing Attacks*, *EM Attacks* and *Optical Attacks* [12]. The observation of the emissions of a device can be seen as a *non-invasive* technique to gather side-channel information. The disassembly, in this case, is not required and the attack, generally, does not leave any trace on the chip package. For instance, when acquiring power-related information, probes may be attached externally to the device, generally on its external pins or on the nearby PCB traces. In other cases, such as electromagnetic observations, probes simply need to be placed in the proximity of the package without the need of physical contact, leaving absolutely no trace on the chip nor on the device enclosure. This goes in opposition to what happens with *invasive* physical attacks, such as *microprobing* and *hardware reverse engineering*, in which the chip's die is extracted from its package (decapped) before being analyzed with very expensive instrumentation like scanning electron microscopes (SEMs).

In general, no real circuit is immune to *passive* side-channel analysis: physical emissions are beyond any processor architecture or higher level specifications. They can be reduced

and masked as much as to make an attack pointless, leading an attacker in spending more than it can retrieve from the attack itself. However, they cannot be prevented from happening, as they are intrinsically tied to the normal usage of a circuit.

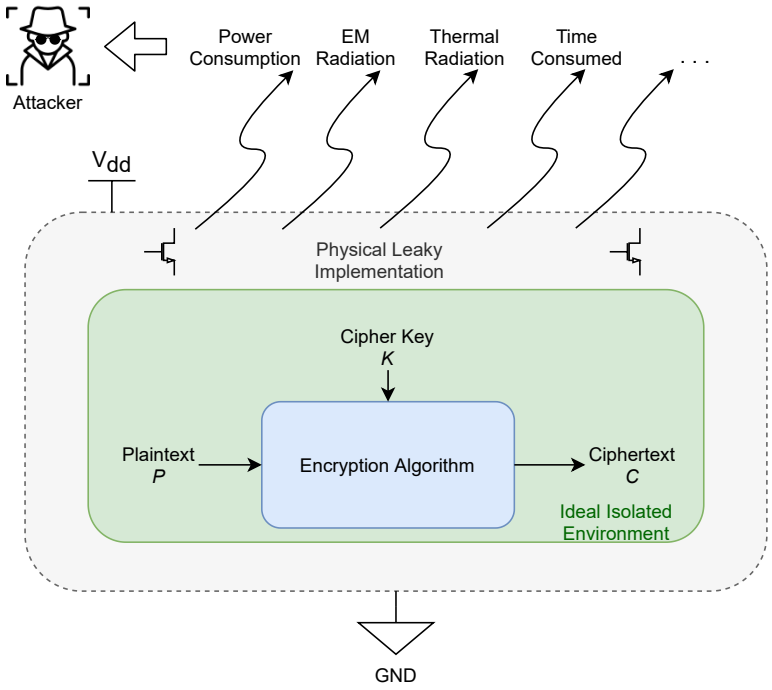

Figure 3.1: Logical representation of the side-channel attack principles.

3.2 Power Analysis

This thesis concentrates on *passive* side-channel analysis, specifically targeting *power attacks*. This methodology was firstly exposed in 1998 by *Paul Kocher*, *Joshua Jaffe* and *Benjamin Jun* with their widely known publication "Differential Power Analysis" [13]. This work laid the foundation for power-related side-channel attacks on computing systems, highlighting the importance of hardware as the backbone of computer system security and opening up a new chapter of exploration for cryptanalysis. Although Kocher's paper demonstrate the ability of power side-channel attacks to overcome the `DES` implementation, the overall methodology and its general effectiveness have been proven multiple times against a multitude of different deployments and ciphers.

Power analysis leverages the fact that the power consumed by a circuit varies according to the activity of its individual transistors and other components. Such transistors (mainly MOSFETs; FinFETs in more recent technologies) act as voltage-controlled switches, relying on the accumulation of electric charges across their terminals. The continuous toggling activity to which a device is subjected implies a continuous charge and discharge of these intrinsic capacitances present in the MOSes structures. This motion of electric charge (current) translates itself in power consumption and electromagnetic emissions, two parameters that, if properly observed, can disclose important "clues" on the data treated internally by a microprocessor.

It is important to highlight that different operations lead to different energy consumption profiles: an idle device consumes much less energy than a device performing a cryptographic operation (such operations are, in general, quite demanding in terms

of computational resources, and thus require non-negligible energy needs). In summary, the power consumption of an integrated circuit or a larger device reflects the aggregate activity of its individual elements [5]. Similarly, on a much finer grain, the toggling activity of transistors depends on the data processed by the circuit in a certain moment in time. To better indicate what is the main goal of the power observations carried by an attacker, we can **logically** distinguish among *operation-dependent* and *data-dependent* power consumption observations. The first relates to coarse grain observations: the attacker is interested in finding *what* is the operation computed by the processor in a certain interval of time, not *how* the operation is specifically manipulating the data. On the other hand, *data-dependent* observations operate on a much finer grain: the attacker is interested in finding, in the specific, *how* the **power trace** relates with the data handled by the processor [1]. Clearly, the first approach will be the one used by the attacker in the very first phases of the analysis: the goal is to gather as much information as possible about the executing conditions of the device. For instance, the attacker has to find, by simply observing the captured power trace(s), what is the time interval in which the processor is executing the main cryptographic operation, discarding other non-targeted computations. Once the main operation has been localized, the attacker switches to a data-dependent observation of the power consumption: the goal now is to carefully observe a specific time interval (where the cryptographic operation is supposed to happen) and relate the power consumption to the data handled by the device. This last step is what will allow the attacker to retrieve the internal information treated by the cipher, such as the round keys used. It is evident that the second observation model is way more complex that the first one. While an operation-dependent observation tries to relate a certain computation to a time interval in the captured trace (hence, trying to find a reference on the x axis), a data-dependent observation tries to relate each voltage sample to a specific data bus transaction (thus, primarily observing what happens on the y axis).

As showed in the next sections, operation-dependent observations are carried on mostly in the case of *Simple Power Analysis* (or *SPA*), whereas data-dependent observations are reserved to the case of *Differential* and *Correlation Power Analysis* (or *DPA* and *CPA*, respectively). It is extremely important to highlight the fact that, while an operation-dependent observation can be done even with a single power trace, a data-dependent analysis is not possible if not by capturing a multitude of traces to be examined at a later time using statistical methods (as no manual observation is feasible in this case).

On a more physical level, it is important to highlight that the total amount of power consumed by a digital circuit is divided into two major components, depending on the nature of the consumption itself: **static power** and **dynamic power** 3.1. Static power indicates the amount of energy consumed whenever the circuit is powered on, independently on the toggling activity and on the operating frequency: a device running idle or at its full performance consumes roughly the same amount of static power. On the other hand, dynamic power indicates the amount of energy consumed due to the switching activity of the transistors: in this case, the power consumption varies depending on the frequency at which the transistors switch their state. Dynamic power can be further divided into two components 3.2: the **short-circuit power** and the **switching power**. The first component corresponds to the power consumed during the switching transient in a CMOS circuit: it is considered negligible in most of the cases due to its reduced impact with respect to the switching power. The second component depends

[1] With the term "**trace**" we indicate a collection of sample points captured during the entire (or partial) activity related to a certain operation performed by the device under attack. In particular, a "**power trace**" can be represented as an interpolated curve composed of many sample points, each of which corresponds to a specific quantized voltage value.

on the circuit capacitance C, on the clock frequency f_{clk}, on the supply voltage V_{dd} and finally on the switching activity E_{sw}, which represents the probability of a transistor to toggle its state.

$$P_{tot} = \underbrace{P_{leak}}_{P_{static}} + \underbrace{P_{sc} + P_{sw}}_{P_{dynamic}} \tag{3.1}$$

$$P_{static} = I_{leak} V_{dd}, \qquad P_{dynamic} = P_{sc} + P_{sw} \approx P_{sw} = \frac{1}{2} E_{sw} f_{clk} C V_{dd}^2 \tag{3.2}$$

Every power attack leverages the observation of the *dynamic power* component of a circuit total power consumption: the *static power*, given its constant nature, does not reflect any toggling activity, therefore does not relates to the actual information propagation happening within the circuit. On the other hand, by observing the dynamic power the attacker is able to correlate, both qualitatively and quantitatively, the power consumption to the stream of information processed by the microprocessor.

In a real scenario, the attacker that inserts the instrumentation probes on the power supply pins of the chip will observe the total power consumption, namely the sum of both static and dynamic components. However, this poses no problem regarding the ultimate goal of the attacker: as with every variable signal, it is sufficient to remove the constant component (the static power) from the signal to obtain only the variable one (the dynamic power).

3.2.1 Attack Instrumentation and Circuit Configuration

To carry out a power attack full access to the device to be attacked is required. For the purposes of a cryptanalyst or penetration tester, the device can be provided directly by the company that has an interest in the results of the attack, in order to assess the security of the product or to find criticalities in its implementation. In the case of a malicious actor, the access to the device can be obtained by simple purchase (the information extracted could be worth much more than the cost of the product) or directly by stealing it. The access to the device due to temporary inattention of the respective owner is unlikely given the time and resources required to build some prior knowledge of the device under attack (e.g., to locate the main cryptographic chip and its power pins in the PCB). It should also be noted that although the required instrumentation is not overly expensive, it cannot be considered easily transportable. In fact, the attacker needs two mandatory tools to first capture the side-channel information and, at a later stage, to process the captured data. These two correspond in most (if not all) cases, to an oscilloscope and a personal computer. In addition, some low-cost off-the-shelf electronics are also needed, as well as an entry-level electronic knowledge. Finally, in some cases, additional tools such as a soldering station may be needed to be able to remove any unwanted components from the device's PCB. In section 3.7 new powerful instruments are presented, already available for purchase and capable of partially easing the problems listed above.

The power consumption of the device can be expressed as:

$$P_{device} = V_{dd} I_{device} \tag{3.3}$$

Since the supply voltage V_{dd} is constant, the variations of the device's power consumption can only be appreciated by observing the current absorbed I_{device}. However, no oscilloscope is able to relate a current variation to the time domain: any trace represented by the instrument corresponds to a time-voltage relation. Therefore, it is mandatory to include the device under attack within an adapting circuit able to "translate" the variable

current consumption into a variable voltage, to be observed with the oscilloscope probe. Such circuit, called *current sensing circuit*, is extremely simple as it consists of a single shunt resistor, leveraging the basics of *Ohm's* law [2]. A shunt resistor is a resistor of very low but accurately known value. It is placed in series to the device, so that virtually all of the current to be measured flows through it. The low value of the resistance allows to measure the resulting voltage drop without significantly interfering with the rest of the circuit (meaning that the device still receives roughly the same supply voltage as before and that the current component due to the shunt resistor is negligible). The shunt resistor can be placed in a high-side or low-side configuration with respect to the device: the position does not significantly impact the voltage measures. In this thesis, an high-side placement of the shunt resistor is considered, as pictured in Fig. 3.2.

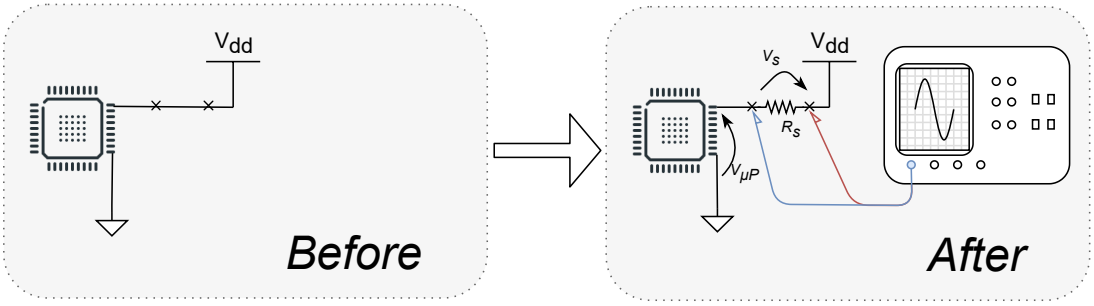

Figure 3.2: Implementation of a current sensing circuit with a high-side shunt resistor.

To implement the overall circuit the attacker has to firstly locate the chip under attack and find its main power supply pins (possibly using the related datasheet, assuming it is publicly available). Once the pins located, the attacker has to find a way to insert the shunt resistor. Some manipulation of the PCB is required, such as desoldering the V_{dd} pin and soldering the shunt in place. If the resistor cannot be inserted (e.g., if a device uses an internal battery), the device's internal resistance is often sufficient [5]. Once the circuit completed, the attacker simply needs to connect the oscilloscope probe to the shunt resistor terminals. The corresponding voltage drop is measured: each time the current absorption increases due to heavier computational requirements the voltage drop across the shunt resistor increases in turn. Therefore, positive spikes on the traces captured correspond to spikes in the current consumption. Conversely, negative spikes are related to a sudden decrease in the power consumption (the current in the sensing circuit lowers so as the measured voltage drop). The probe must be AC-coupled: this allows to cut-off the entire DC component of the signal, keeping only the voltage variations around the rejected mean value.

In general, as recalled by Kocher itself, measurements taken closer to the cryptographic component will generally be of better quality. In the case the chip has multiple power supply pins, the attacker should measure the voltage on the one that directly powers the targeted internal circuitry. The removal of decoupling capacitors is beneficial as well, as they tend to eliminate rapid voltage variations, working against the measurements goals. In addition, it is recommended to use quality probes and interconnects, possibly shielded to avoid the influence of noise from the surrounding environment. Finally, following the same noise-reducing principle, the attacker should use an external bench-grade power supply to power the device under attack with the cleanest possible supply voltage: the measured

[2] *Ohm's* law points out that the voltage drop across a resistor is linearly proportional to the current that flows through it: $V_R = RI$.

voltage variations shall come from the device, not from the external environment.

As a consequence of these arrangements, the attacker can reduce the total number of traces to be collected. A reduced number of traces is especially useful when considering the heavy statistical computations required for both DPA and CPA: less traces means less computational effort which leads to a faster attack and, potentially, to a better final result (see 3.4 and 3.5).

3.3 Simple Power Analysis (SPA)

Simple Power Analysis is the first technique an attacker can leverage to have some initial understandings of the device operations, basically corresponding to what previously referred as *operation-related* observations. As defined by Kocher, Jaffe and Jun, SPA is "*a technique that involves directly interpreting power consumption measurements collected during cryptographic operations. SPA can yield information about a device's operation as well as key material.*" [13]. The analysis exploits qualitative observations of the power consumption, often done visually by the attacker itself. The goal is to leverage relevant power fluctuations and macro-characteristics to identify the microprocessor activity. By investigating the overall shape of a single power trace (nothing prevents observing multiple traces), the attacker can recognize the presence of a certain algorithm, possibly distinguishing its individual internal operations (see example 3.3.1). In some cases, an in-depth analysis can go to the extent of discerning each single clock cycle of the device under attack, allowing the attacker to recognize and identify the assembly operations performed by the microcontroller, as demonstrated in example 3.3.1.

Given the nature of the observations made with SPA, it is possible to exploit the information obtained from multiple power traces as additional clues for a reverse engineering attack. When stimulated with different inputs, the power leaks emitted by the device can report different paths in the the code flow, highlighting the presence of possible conditional branches. Again, an example of this scenario is reported in 3.3.1.

With SPA, however, it is very unlikely for the attacker to retrieve the data related to a cipher key: data-related observations, in the case of block cipher implementations, are almost unfeasible without relying on statistical computations. In any case, the basic information gathered is of extremely importance for the next steps of the analysis, as it allows the cryptanalyst to understand the device behavior and the overall environment. In summary, SPA provides a very effective and efficient way to obtain the information necessary to solve for the secret keys, therefore, it should not be an overlooked step [5].

3.3.1 SPA: Attack Demonstrations

Used as a tool to identify macro-characteristics of the device power consumption, a carefully crafted SPA attack can also be leveraged to infer even smaller clues about the operations performed by a device.

This section demonstrates what an attacker can deduce by simply observing a power trace captured using a synchronous sampling technique. The capture tool used is a *ChipWhisperer™ Lite* board, connected to the target device via a coaxial SMA probe, whose terminal is connected between the chip itself and the shunt resistor. In our case, the chip is an `ATXMEGA128D4` 8-bit RISC microcontroller, produced by *Atmel*. Additional information about the setup used for this thesis work will be given in 3.7.

Example 1 - Annotating a simple C program The example below demonstrates to what extent it is possible to correlate a certain operation with its power trace. Let us assume being in possession of a given `C` program, executed by the `XMEGA` target. Let us also assume the knowledge of the related assembly code.
The `C` code is the following:

```
int main(void)
{
    volatile long int A = 0x2BAA;
    while(1)
    {
        A *= 2;
        __asm__("nop");
        __asm__("nop");
    }
}
```

The related `AVR` assembly code is the following:

```
        A *= 2;
79c:        89 81   ldd r24, Y+1 ; 0x01
79e:        9a 81   ldd r25, Y+2 ; 0x02
7a0:        ab 81   ldd r26, Y+3 ; 0x03
7a2:        bc 81   ldd r27, Y+4 ; 0x04
7a4:        88 0f   add r24, r24
7a6:        99 1f   adc r25, r25
7a8:        aa 1f   adc r26, r26
7aa:        bb 1f   adc r27, r27
7ac:        89 83   std Y+1, r24 ; 0x01
7ae:        9a 83   std Y+2, r25 ; 0x02
7b0:        ab 83   std Y+3, r26 ; 0x03
7b2:        bc 83   std Y+4, r27 ; 0x04
    __asm__("nop");
7b4:        00 00   nop
    __asm__("nop");
7b6:        00 00   nop
while (1)
7b8:        91 cf   rjmp .-222  ; 0x6dc <main+0x60>
```

As it is possible to see by comparing the two listings, the `AVR` compiler translates the `C` code into 15 assembly operations: 12 for the main computation, 2 for the two `NOP`s and 1 for the `while(1)` instruction. By referring to the datasheet related to the `XMEGA` family of microcontrollers [25] we obtain that:

- the `ldd` instructions take 3 clock cycles each;
- the `add` and `adc` instructions take 1 clock cycle each;
- the `std` instructions take 2 clock cycles each;
- the `rjmp` instruction takes 2 clock cycles.

Fig. 3.3 shows two annotated graphs for two different captured power traces. Both graphs represent the end result of an SPA attack: the observed power leakage was enough to obtain important information about the computations done by the device. Namely, the bottom graph refers the specific case just mentioned (identifiable by the presence of two `NOP`s instructions). The image proves that it is possible to locate each assembly instruction by observing its related spike/power sample in the trace graph. In detail, each major spike in the image represents the very first sample among the four collected for each clock cycle of the target (see Fig. 3.16). To prove the correctness of the analysis let us remove one of the two `NOP`s and compare the new trace with the previous one: if the new trace obtained shows one spike less then our annotations are correct. The graph corresponding to this test is the top one in the same image 3.3. The only major difference in the collected samples consists in the lacking of the peak related to the `NOP` instruction we just removed, hence confirming our annotations.

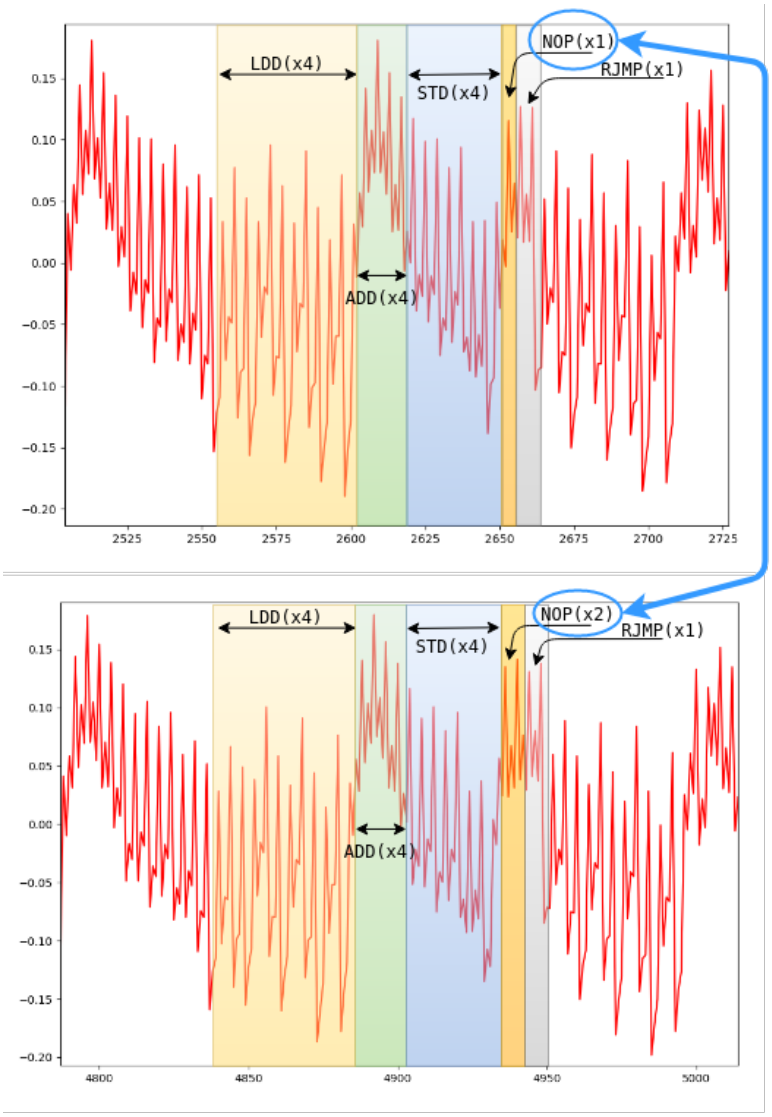

Figure 3.3: Visual analysis of two slightly different power traces: the second one includes an additional `NOP` instruction, denoted by an additional spike in the trace. For both traces, the attacker is able to group the various peaks into the corresponding sequences of assembly instructions.

Example 2 - Annotating AES-128 Fig. 3.4 reports the observations and consequent annotations done on a power trace referring the execution of `AES-128`. A simple qualitative observation is sufficient for an experienced attacker to infer the operations executed: the various colored groups refer, in fact, to the different operations that compose the `AES-128` algorithm, such as the `ShiftRows` and `MixColumns` operation. The case of the `ShiftRows` step is striking: the presence of 3 main spikes clearly indicates the 3 rotations done on the State matrix (the first row of the State is kept as is). Additionally, a closer look to the the main peaks present in both the `AddRoundKey` and `SubBytes` sets can give a further clue for the attacker: the fact that each group contains exactly 16 main spikes, each related to a different byte of the State, confirms the presence of a 128-bit block cipher.

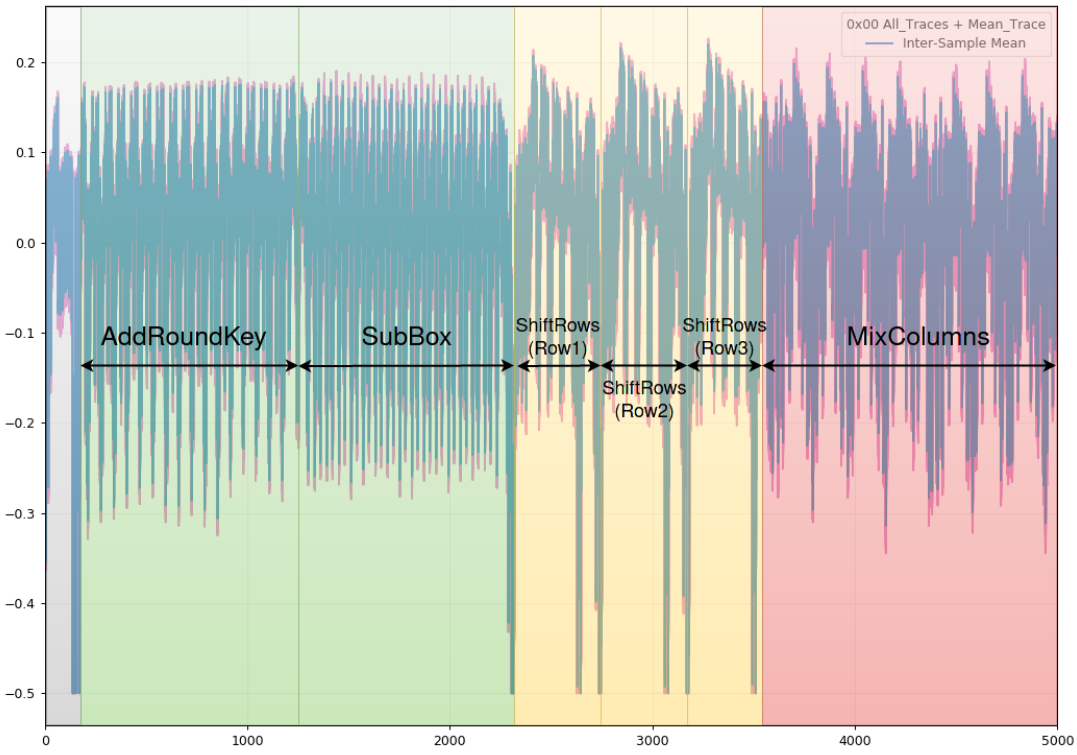

Figure 3.4: Annotations of the *key Whitening* operation followed by the very first round of `AES-128`.

Example 3 - Highlighting Code Flow Differences Superimposing and observing multiple traces can highlight differences in the flow of the code executed by the device. Each trace captured is related to a different execution of the target device: similarly to a chosen plaintext attack, the attacker can submit a different set of inputs per execution. In this example, the `C` code executed by the device completes a password check by sequentially checking each submitted character: if the character is different from the expected one then the loop that iterates on all the characters stops iterating, jumping on the code section in charge of emitting an error code.
The `C` code of the example is the following:

```c
// Hardcode expected password
char passwd[32];
char correct_passwd[] = "h0px3";
//Get user password
my_puts("Please enter password to continue: ");
my_read(passwd, 32);
uint8_t passbad = 0;
trigger_high(); // Trigger for trace capture

// sizeof(correct_passwd) == 6
for (uint8_t i = 0; i < sizeof(correct_passwd); i++) {
    if (correct_passwd[i] != passwd[i]) {
        passbad = 1;
        break;
    }
}
if (passbad) {
    my_puts("PASSWORD FAIL\n");
    led_error(1);
}
else {
    my_puts("Access granted, Welcome!\n");
    led_ok(1);
}
```

Figure 3.5 shows the alignment and superposition of 36 different power traces, each corresponding to a different possible input character (assuming the password cannot contain characters out of this set). The superposition highlights the presence of different paths in the code flow: in this case, character "`h`" leads to a unique power consumption profile, proving the presence of a conditional branch. Since all possible character have been tested, the attacker evinces that "`h`" must be a character of the password. The attacker can now iterate the same attack for each of the remaining characters: eventually, the entire password will be revealed and the password check cracked.

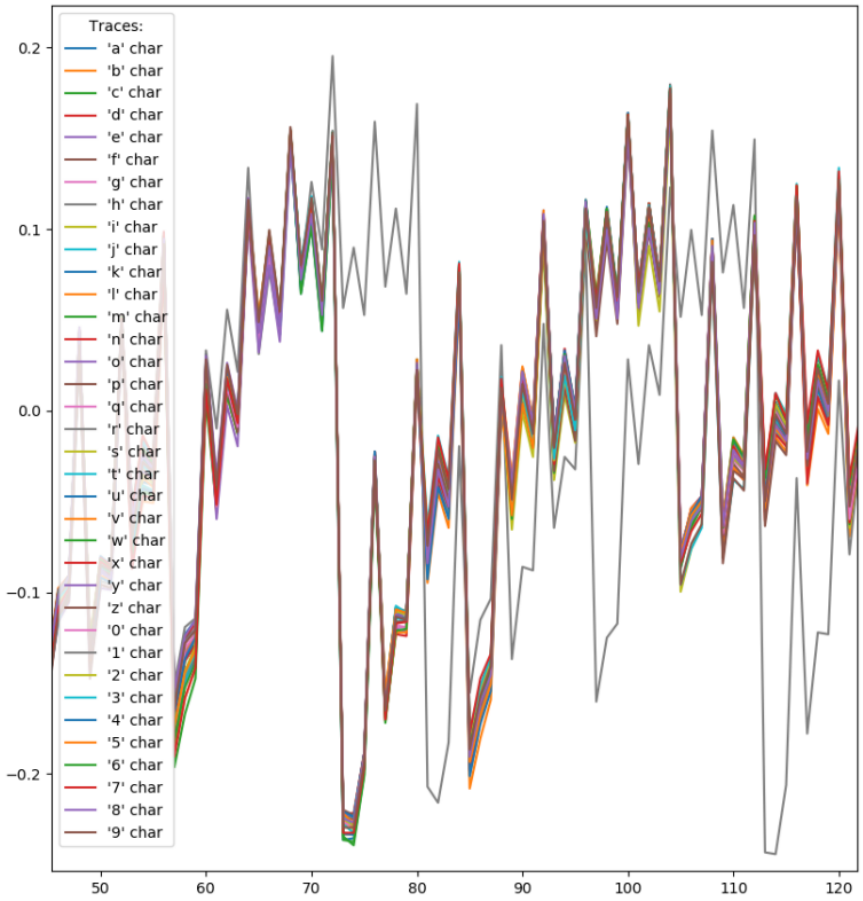

Figure 3.5: SPA used to highlight the presence of conditional branches in the code flow.

3.4 Differential Power Analysis (DPA)

Differential Power Analysis (or *DPA*) is the first of the two main statistics-based methods regarding the class of power side-channel attacks. DPA allows the attacker to conduct *data-related* observations, correlating the power consumption measured in a certain instant in time to the data handled by the microprocessor under attack. In contrast with SPA, statistical analysis like DPA normally require a large amount of power traces to achieve a key recovery. In some cases, the collected traces may need an additional conditioning step before the main statistical computations. For instance, the attacker may want to remove unwanted samples (which may refer to operations preceding or following the targeted one) and to remove possible alignment errors. The temporal alignment of the collected traces can in fact significantly improve the efficiency of the attack and the quality of the results obtained. Trace alignment becomes a mandatory step in all those real-world scenarios where the trace capture is launched in slightly different time instants. The misalignment can also be caused by the presence of jitter between the sampling clock and the device clock: the use (if possible) of synchronous sampling solves this problem (see Fig. 3.8).

DPA bases its observations on the existence of an intermediate value within the block cipher data that, if considered in the ensemble of the implementation environment, can be physically correlated to a specific contribute to the power consumption. This intermediate value must also be the result of an operation involving the plaintext (ciphertext) and (part of) the key.

These two "rules" are based on the (correct) assumption that every single bit transferred on the data bus causes a certain amount of power consumption. The interconnects that form the parallel data bus between the register file and the processor core can be (roughly) seen as load capacitances that are charged/discharged at each data transfer. As shown in Fig. 3.7, different bit transfers imply different spikes on the chip power rail: it follows that if more bus lines have to be charged, more current will be needed to charge the related parasitic capacitances. Differential power analysis exploits the correlation between the value assumed by a single transferred target bit and the consequent presence of a spike in the microcontroller power rail. If the bit is `1`, we assume the presence of a spike in the captured power trace, while if the bit is `0`, we assume its absence. In most of the cases the bit under observation makes up the result of the `XOR` operation between the plaintext and the cipher key, so to satisfy the rules previously mentioned. As a tacit convention, generally this bit coincides with the LSB: in reality every bit position can be considered.

This assumption is what defines the most common policy of a DPA *prediction function*. This function defines the criterion to be applied when correlating the power consumption to the data processed internally by the block cipher implementation. Several policies may be defined: some may lead to better results than others, depending on the physical environment in which the attack is applied. Moreover, several prediction functions can be used in the same attack: the different results obtained can be compared with each other to find trade-offs between the quality of the results and the computational complexity required to obtain them.

The following step of the attack involves the so called *selection function*. Its goal is to apply the policy dictated by the prediction function to the collected power traces. Assuming the most common policy just mentioned, the selection function evaluates and separates the traces into two distinct sets: the first is associated to the target bit having value `1`, while the second relates to the target bit having value `0`. To partition the various traces, the selection function must leverage some partial information coming from the block cipher: the intermediate value. However, considering the nature of the attacker's target operating environment, no encrypted data is directly observable apart

from side-channel emissions. In any case, if the attacker were able to extract intermediate data directly from the device, it would make no sense to attempt a side-channel attack in the first place.

To find this intermediate value, a reduced replica of the block cipher under attack is built. This is possible thanks to the open source nature and public availability of the specifications of modern block ciphers, which all follow *Kerckhoffs' Principle*.

In many real-world scenarios involving `AES-128`, for instance, the model built by the attacker replicates a very small portion of the block cipher algorithm, formed by combining the Key Whitening operation (`AddRoundKey`) and the subsequent `SubBytes` step (the one related to the first round). The result obtained with this ideal replica is then masked in order to isolate the targeted bit. The selection function will then assign each trace to one set or the other according to the bit's value. The choice of the `AddRoundKey` and `SubBytes` operations for the model is not accidental: as mentioned in 2.2.2, only the 128-bit variant of `AES` uses the entire main secret as round key for the very first `AddRoundKey` operation. Therefore, by targeting the Key Whitening step, the attacker can expect to retrieve the main secret immediately, without having to analyse multiple rounds or to resort to the key schedule algorithm.

Once the partitioning phase has evaluated the entire trace collection, two representative average traces are computed for both of the two sets. Finally, the difference among these traces is determined. If the choice of which trace is assigned to each set is uncorrelated to the measurements contained in the traces, the difference in the sets' averages will approach zero as the number of traces increases. Otherwise, if the partitioning into subsets is correlated to the trace measurements, the averages will approach a non-zero value [5]. Given enough traces, extremely tiny correlations can be isolated no matter how much noise is present in the measurements. The presence of non-zero values in the differential trace indicates that the partitioning phase (and the policy applied) correctly assumed the correlation between the power samples and the value of the observed intermediate bit. Better policies better highlight the differences between the two sets: having a "good" prediction means that the attacker is able to correctly assume the correlation between the device power consumption and the data handled by it. Consequently, a good prediction function provides better results in lower computational time.

In Fig. 3.6, provided by Kocher in [5], the authors show two probability distribution functions of measurements obtained at a certain offset in a set of collected traces. The two Gaussian curves have been obtained by partitioning the entire trace collection in two main sets following the exact same selection function presented above. The distance between the two curves is due to the partitioning applied: a good prediction policy better distances the two curves, as the partitioning better separates the traces collected. As a consequence, a higher distance between the two Gaussians indicates the applied prediction function leads to better quality of the final results, meaning the attacker can retrieve a larger portion of the key in less time. As highlighted by the authors, the left-hand side curve (LSB @ "`1`") presents a power consumption with a lower mean with respect to the curve on the right-hand side (LSB @ "`0`"). This counterintuitive scenario is due to the placement of inverters by logic synthesis tools and other design details, de facto making it possible for either value to consume more power. In any case, what matters is that these two distributions are significantly different, demonstrating that the power consumption is statistically correlated to the LSB of the `S-Box` output [5].

3.4.1 DPA: Attacking an `AES-128` software implementation

This example illustrates the main operations that characterize a DPA attack against `AES-128`: for the sake of brevity, let us assume the attacker has already collected a number

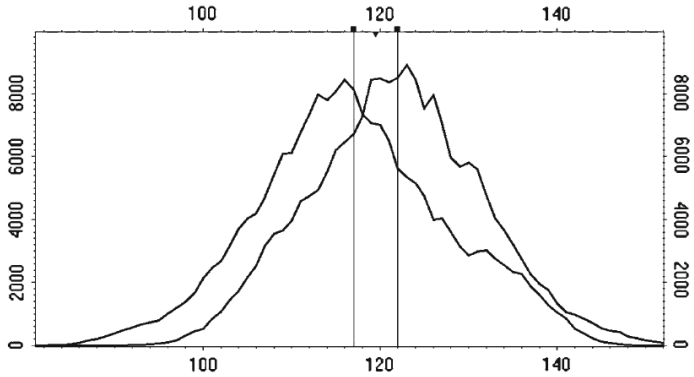

Fig. 5 Distributions of power consumption measurements for traces with the LSB of the output of the first S-box being 1 (left) and 0 (right)

Figure 3.6: Visual representation of the effects due to partitioning: the higher the distance between the two curves the better the prediction function [5].

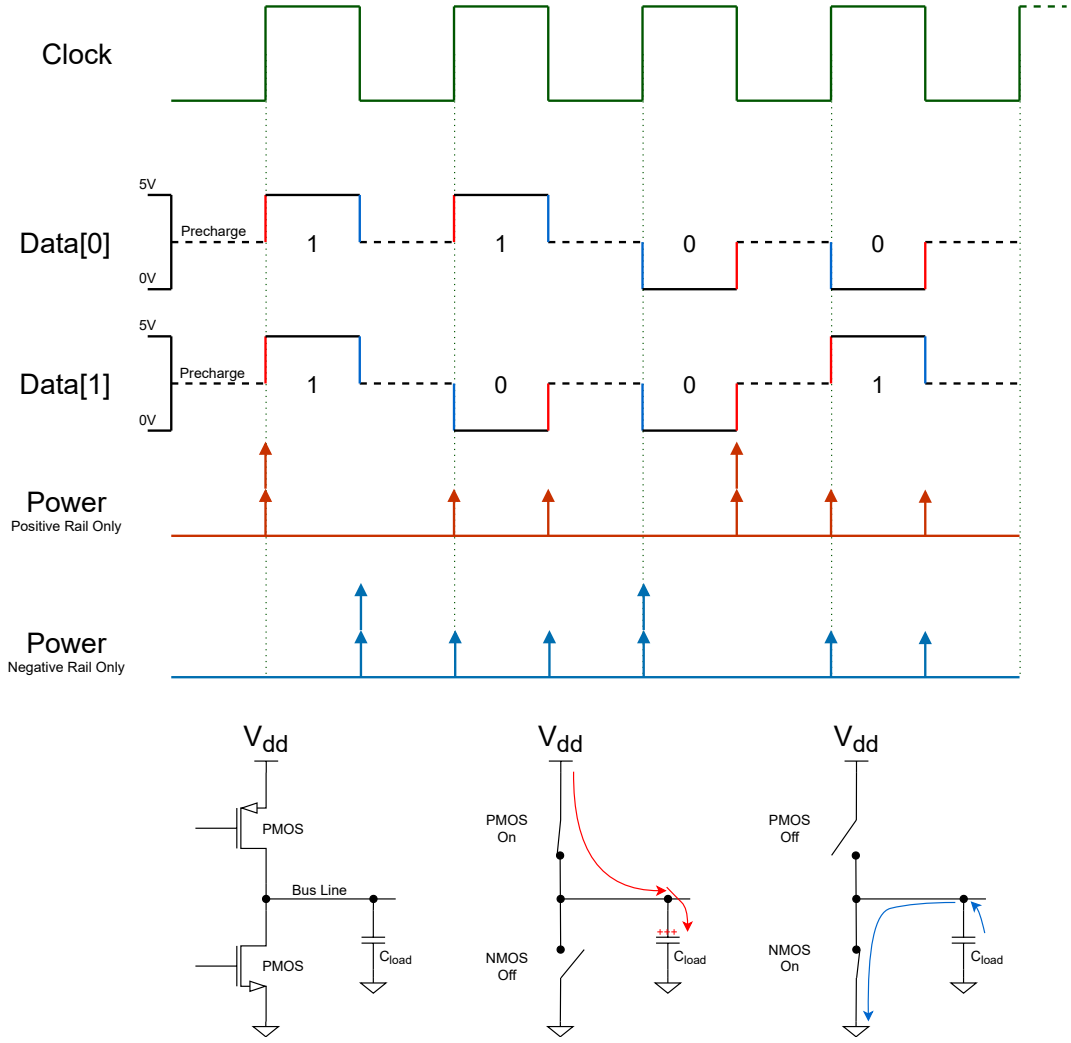

Figure 3.7: Visual representation of the effect of data bus transfers on the power rails, assuming a microcontroller with precharge logic.

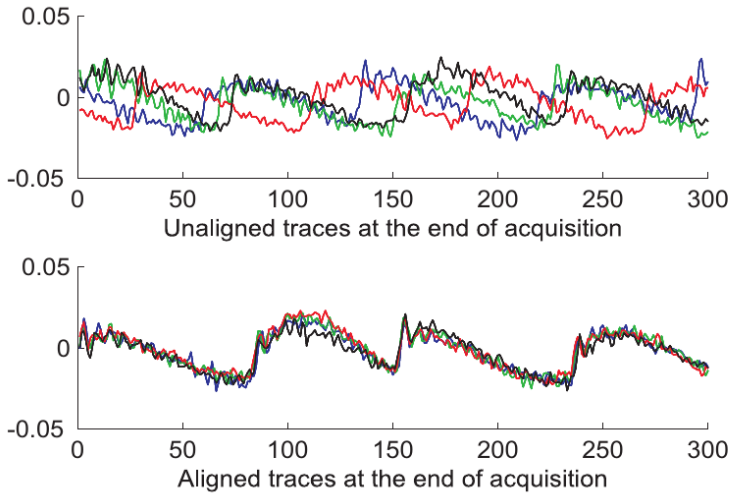

Figure 3.8: Traces affected by high clock frequency jitter before and after trace alignment [6].

N of different [plaintext - trace] pairs ($[P_i, T_i]$). Each power trace is composed by S sample points: each point corresponds to a quantized voltage measure.

In a real scenario, a complete DPA attack involves a multitude of iterations, primarily due to:

- The number of `[plaintext - trace]` pairs collected;
- The number of samples that make up each power trace;
- The number of bytes of the key (out of 16) the attacker wants to retrieve;
- The number of iterations necessary to recover each byte of the key ($2^8 = 256$ iterations needed per byte).

A complete DPA attack is carried on using an iterative *Divide et Impera* approach, meaning that all the 16 bytes that compose an `AES-128` key are addressed one at a time. This allows to break the attack in 16 different and sequential iterations: once the b-th key byte has been retrieved, the attack proceeds by cracking the following $(b+1)$-th. This approach allows the attacker to try every possible value a key byte can assume (from `0x00` to `0xFF`), de facto implementing a small (and feasible) exhaustive search on $n = 2^8 = 256$ values.

Each of the 256 key candidates K_n is fed into the model built by the attacker (see equation 3.4) to derive the value of $I_{i,n}$, evaluating each $[P_i, T_i]$ pair. Thereafter, the selection function selects the targeted bit from the intermediate State: Let us assume, for this example, the targeted bit corresponds to the LSB of $I_{i,n}$. Each trace is assigned to one of two subsets, depending on whether the selection function result is "0" or "1" [5]. The difference of the subsets' averages is then examined (also called *Difference of Means* or *DoM*). If the value of the `S-Box` output bit predicted by the selection function has even a tiny correlation to the power traces, the DPA test will show spikes indicating that the key guess K_n is correct. For each wrong candidate K_n, the predicted values of $I_{i,n}$ will be (largely) unrelated to any data being processed by the target device, and the DPA test will not be (or will be much less) statistically significant.

$$I_{i,n} = \texttt{S-Box}[P_{i,n} \oplus K_n] \tag{3.4}$$

byte	#0	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15
	0x2B	0x7E	0x15	0x16	0x28	0xAE	0xD2	0xA6	0xAB	0xF7	0x15	0x88	0x09	0xCF	0x4F	0x3C

Table 3.1: The secret key to used for this thesis work.

In most real-case scenarios, the differential traces computed for the wrong key guesses may still show some nonzero peaks of correlation. Their presence is mainly due to two reasons: the contribute due to measurement noise and the lack of an ideal independence among the target intermediate value produced by the correct key and the target intermediate value produced by the wrong key candidates. As a consequence, the attack may not succeed for each of the 16 iterations to be completed: when analyzing a specific byte some of the 256 guesses may return better correlations although not being the correct ones due to the above noise contributions. In any case, the attacker can easily build a list of all the key candidates that returned a higher correlation than the others, ranking all of them based on the differential score they are associated to. This score can have multiple definitions: in the most common case it corresponds to the value recorded by the highest peak in the absolute differential trace $|\Delta T|$, indicating the highest correlation registered for the current key guess.

The above problem is solved by increasing the number of processed power traces: both noise contributions become less significant while the correlations for the correct guess become more prominent. Adding more traces to be processed leads to an additional computational load and to longer attacks. Hence, it is advisable to start with a reduced number of traces and follow an incremental approach until the correct key is discovered. A proper attack stops as soon as the result obtained is satisfactory, avoiding calculations that are not strictly necessary.

In Fig. 3.9 shows the correlation traces produced by a full attack on the 16 bytes of the secret key in 3.1. The graph is constructed by overlaying the entire collection of power traces (it is possible to recognize the same pattern that characterizes the profile of `AES-128`, see Fig. 3.4). On top of these traces, 16 different absolute differential traces are superimposed, each of which is associated with the key guess having the same value as the corresponding byte of the secret key. As can be seen from the spikes registered on each differential trace, correlation peaks at exactly the position where each `S-Box` lookup is performed. For example, the first trace, corresponding to the key byte `0x2B`, peaks just after the completion of the `AddRoundKey` step, on the first `SubBytes` lookup. This effectively confirms the correctness of the applied policy, which assumes the presence of a correlation between the LSB of the `S-Box` output result and the energy consumption. Even higher correlation peaks can be observed when the `MixColumns` step is executed: due to the nature of `AES`, these correlations are generally rendered undetectable after 2 rounds.

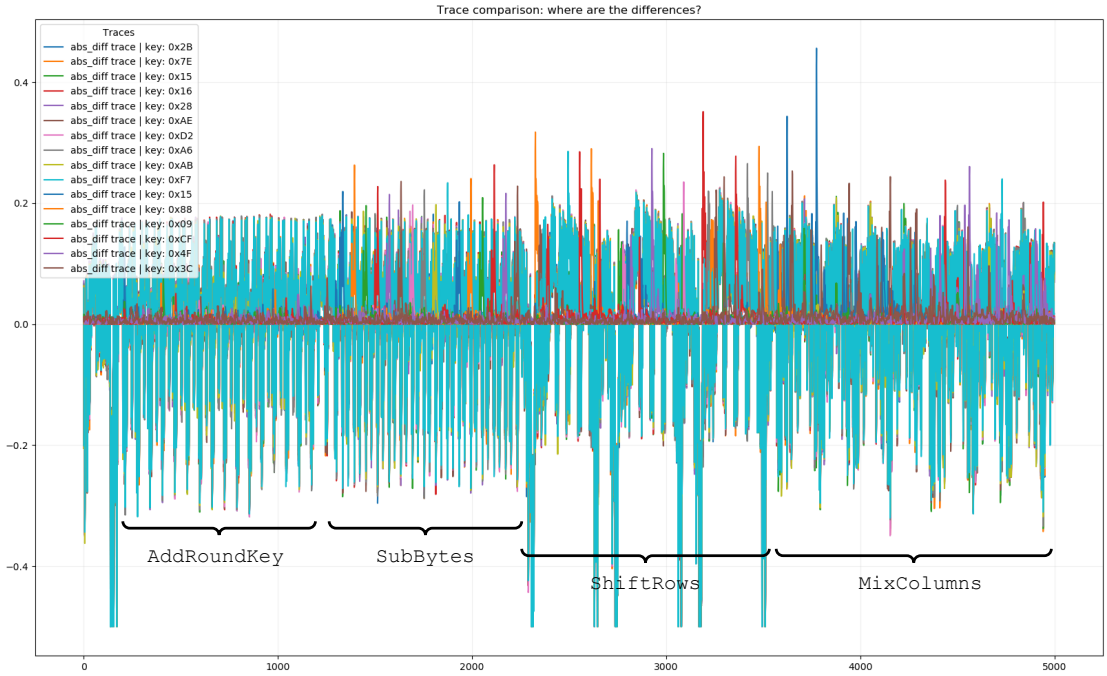

Figure 3.9: Superposition between the captured power traces and the (magnified) absolute differential traces $|\Delta T|$ (one differential trace for each byte of the key).

3.5 Correlation Power Analysis (CPA)

Correlation Power Analysis (or *CPA*) is the second statistical technique able to retrieve the entire content of a secret key by leveraging the collection of multiple power traces. It is based on similar principles as DPA, to the point of being, in some cases, considered as a subclass attack of it. However, correlation power analysis takes a step further in the attempt to correlate the data internally treated by the physical implementation with the resulting power consumption. For the purposes of a differential power attack, it is sufficient to prove *the presence* of a power consumption contribution whenever certain conditions on an intermediate State are verified device side, without being interested neither in its relative nor absolute value. The assumption is made *a priori* and later statistically verified provided that a sufficient number of power traces has been collected.

In the case of CPA, the main assumption relies on the fact that each single bit that makes up the intermediate value brings a specific and relative contribute to the overall power consumption. Therefore, the focus of the attack is on the cumulative contribution caused by the toggling bits. To better understand this rationale, let us refer again to Fig. 3.7. In the specific, CPA relies on the fact that each single bus line and the corresponding parasitic capacitance require a certain amount of charge whenever its state toggles. This translates in a specific current absorption and in a corresponding power spike on the power supply rails of the microprocessor under attack.

3.5.1 The Leakage Model

CPA defines the so called *leakage model*: a representation of the device leakage that depends on the value of (or a function of) one or more intermediates in the cryptographic calculation. [5]. In summary, CPA and the leakage model extend the concept applied by the DPA's selection function from a single-bit policy to a multi-bit one. CPA is most

effective in white-box analysis where the device leakage model is known by the attacker: if the attack leads to poor results the cause generally relates either to a low-leaky device implementation or to a poor assumption of the leakage model [5]. Similarly to the *selection function* in the case of DPA, the attacker can test multiple *leakage models* to assess which one leads to better results, trading off the additional flexibility to the time required to end the attack.

The Hamming weight (HW) Model In many applications, the leakage model is defined as the Hamming weight of a multi-bit value $I_{i,b}$ in a register (or on a bus), meaning that $HW(I_{i,b})$ is assumed to roughly estimate the power consumption [3]. For instance, assuming an 8-bit microcontroller having an 8-bit wide data bus, the byte `0x05` shall consume less power than the byte `0xFF`, since the latter has a larger Hamming weight $HW(\texttt{0xFF} = 8)$ than the former $HW(\texttt{0x05} = 2)$.

It is also important to highlight that, in the generic case, if cipher's intermediate value $I_{i,b}$ contains m independent and uniformly distributed bits, the whole word has an average Hamming weight $\mu_H = m/2$ and a variance $\sigma_H^2 = m/4$ [26].

The Hamming Distance (HD) Model The Hamming Distance model, as proposed by [26], can be seen as a generalization of the previous one. While the HW model focuses on the intermediate value assumed by the state $I_{i,b}$ at a specific timing, the HD one focuses on the power consumption related to the number of bits switching their state when transitioning from a value to the successive one. For instance, the transition of the $I_{i,b}$ value from `0x13` to `0x37` denotes an Hamming Distance $HD(\texttt{0x13} \rightarrow \texttt{0x37}) = HD(\texttt{0x13} \oplus \texttt{0x37}) = 2$, since only 2 bits flip during the transition.

The authors in [26] assume the presence of a reference word R considered as the starting state, while calling $I_{i,b}$ the arrival state, therefore defining the generic Hamming Distance as:

$$HD(R \oplus I_{i,n}) \tag{3.5}$$

Like in the previous model, if $I_{i,b}$ is a uniform random variable, so is $R \oplus I_{i,b}$, and $HD(R \oplus I_{i,b})$ has the the same mean $\mu_H = m/2$ and a variance $\sigma_H^2 = m/4$.

The definition given in 3.5 encloses the Hamming weight model, which is obtainable by simply considering $R = 0$. Sometimes the reference state R is systematically 0 due to the presence of pre-charge logic, that clears the bus between each transferred value 3.7.

In general, the presence of a linear relationship between the current consumption and the Hamming models is also assumed. This can be seen as a limitation but, as highlighted by the authors, the linear model fits reality quite well if we consider a chip as a large set of elementary electrical components [26]. Indeed, Fig. 3.11 shows the presence of such linear relationship in our `XMEGA` target board, computed once again using *ChipWhisperer*™.

Both models are not too unrealistic since the bus lines are usually considered as the most consuming elements within a microprocessor/microcontroller. However, power leaks in real worlds scenarios can be complicated. In some devices, transitions from `0` to `1` leak differently than transitions from `1` to `0`. Hamming weight and Hamming distance models treat bits independently but, in real devices, the amplitude of leaks varies significantly for different bits, and may be further modulated by multi-bit effects. For instance, some devices have word-oriented leaks which may be correlated to flag bits such as sign, zero, overflow, or carry [5]. This means that not every device is to be expected of leaking

[3] Remember that $i \in \{0, \ldots, N\}$ where N indicates the number of traces captured, and $b \in 0, \ldots, 15$ indicates the byte position in the plaintext P_i, key K and intermediate state I_i

linearly in the number of bits, as pictured in Fig. 3.10. In this image, taken from [7], the entire collection of traces has been partitioned into 9 sets, each corresponding to a different value of the Hamming weight assumed by a certain intermediate value in a specific time instant. The graph shows 9 different mean traces, each one computed from all the traces of the corresponding set. As it is possible to see, the vertical distance between the traces having $HW = 7$ and $HW = 8$ is way higher than the distance between the trace associated to $HW = 0$ and the one related to $HW = 1$, proving the non linearity of the power consumption contributions.

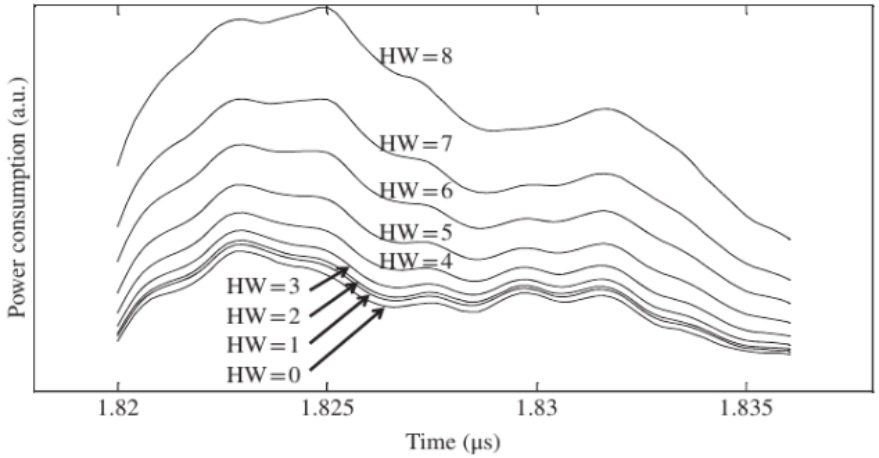

Figure 3.10: Graphical representation of the distance among 9 different mean traces, each representative of the Hamming weight related to its origin set. The relation is not linear. [7]

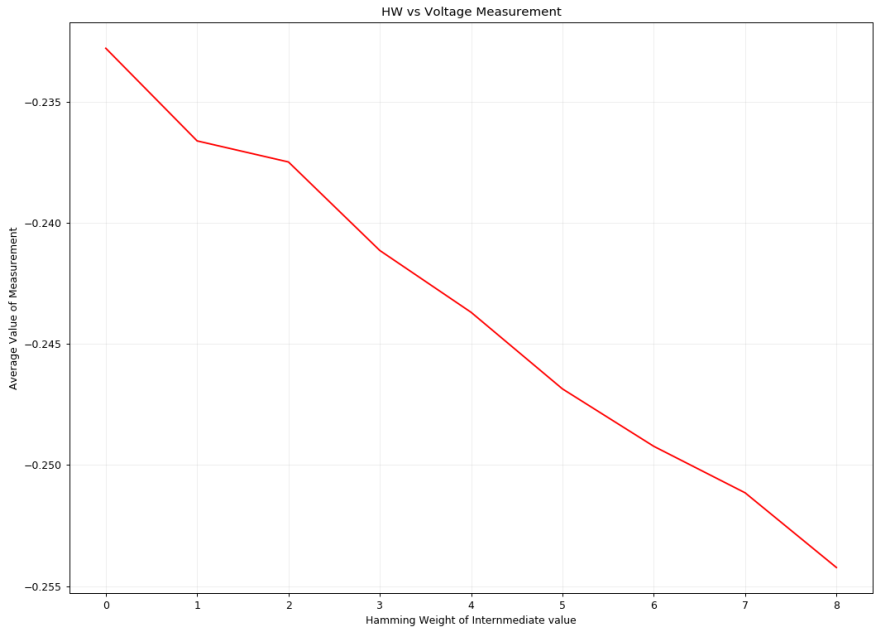

Figure 3.11: Presence of a linear relationship between the current consumption and the Hamming weight of the intermediate values, in the case of the `XMEGA` target device.

3.5.2 Computing the estimated leakage

Assuming the attacker has already formed a collection of power traces, the goal is now to build an estimation of the leakage produced by each intermediate state $I_{i,b}$ leveraging the selected leakage model. This is achieved by building a reduced replica of the block cipher, as done in the case of DPA. The intermediate result returned by the replica corresponds to the same data expected to be internally elaborated by the device implementation. The output of the replica is then fed to the selected leakage model which, assuming the use of the Hamming weight criterion, returns an estimate of the energy consumption $h_{i,b}$.

$$h_{i,b} = HW[I_{i,b}] = HW[\texttt{S-Box}[P_{i,b} \oplus K_b]] \tag{3.6}$$

Equation 3.6 is applied following the same *Divide and Conquer* approach leveraged for DPA: each of the 16 b key bytes to be evaluated carry an additional evaluation on $n = 2^8 = 256$ possible key candidates.

3.5.3 Correlating the estimated leakage to the real power consumption

The HW model assumes that there is a linear dependency between the Hamming weight of the intermediate value and the power consumption, i.e. a correlation. To extract and prove the correlation between the rough estimation of power consumption and the real power traces, a so called *evaluation function* is needed. In most of the cases, this function corresponds to *Pearson's correlation coefficient* ρ [7]. In statistics, the Pearson's correlation coefficient (PCC) is a measure of linear correlation between two sets of data. It is essentially a normalised measurement of the covariance, such that the result always has a value between -1 and 1. As with covariance itself, the measure can only reflect a linear correlation of variables, and ignores many other types of relationship or correlation [27]. Given a pair of random variables (X, Y), the Pearson coefficient ρ can be written as follows:

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma x \sigma y} = \frac{\mathbb{E}[(X - \mu x)(Y - \mu y)]}{\sqrt{\mathbb{E}[(X - \mu x)^2]\mathbb{E}[(Y - \mu y)^2]}} \tag{3.7}$$

where $\mathbb{E}$ is the expectation, μx is the mean of X, μy is the mean of Y and σx, σy are the standard deviations of X and Y, respectively.

As mentioned, the value of $\rho_{X,Y}$ will always be in the range $[-1, 1]$, describing how close the two variables X and Y are related:

- If Y always increases when X increases, it will be 1;
- If Y always decreases when X increases, it will be -1;
- If Y is totally independent of X, it will be 0.

Let us assume the following:

- the attacker has collected N `[plaintext - trace]` pairs, denoted as $[P_i, T_i]$;
- each trace T_i ($i \in 0, \ldots, N$) is formed by S sample points, indexed by the variable $j \in \{0, \ldots, S\}$;
- n indicates the n-th key guess, among all the $n = 2^8 = 256$ possible values a key byte can assume;
- $h_{i,n}$ indicates the power estimate in trace i for key candidate n, as computed in 3.6.

Let us write the PCC in case of a CPA attack as:

$$r_{n,j} = \frac{\sum_{i=1}^{N}[(h_{i,n} - \overline{h_i})(T_{i,n} - \overline{T_i})]}{\sqrt{\sum_{i=1}^{N}[(h_{i,n} - \overline{h_i})^2(T_{i,n} - \overline{T_i})^2]}} \tag{3.8}$$

The following is an alternative form of the equation that can be used for *online* calculations, as it allows the attacker to add one trace at a time without re-computing all of the past data [28]:

$$r_{n,j} = \frac{N\sum_{i=1}^{N}(h_{i,n}T_{i,j}) - \sum_{i=1}^{N} h_{i,n} \sum_{i=1}^{N} T_{i,j}}{\sqrt{[(\sum_{i=1}^{N} h_{i,n})^2 - N\sum_{i=1}^{N} h_{i,n}^2] - [(\sum_{i=1}^{N} T_{i,j})^2 - N\sum_{i=1}^{N} T_{i,j}^2]}} \tag{3.9}$$

By iterating the previous computations on all the possible key guesses, for all the captured traces and for all the samples in a trace, the attacker builds a ranking of the best performing key candidates for the given key byte position b where $b \in \{0, \ldots, 15\}$, sorting them by highest correlation score. To retrieve the entire key, this entire process is iterated 16 times.

Similarly to DPA, the more the traces computed the better the correct key guess will be highlighted with respect to the wrong ones. With a high enough number of traces the correct candidate will "float" up to the top of the ranking: generally, due to the higher efficiency of CPA over DPA, a way lower number of traces is sufficient. An example of the results obtained by attacking `AES-128` can be seen in Fig. 3.12. In Fig. 3.13 and 3.14 we report the superposition of all the 16 correlation traces related to the correct key guesses. The first graph has been obtained with just 50 traces, the second one with 800 traces [4]. The comparison between the two shows how the uncorrelated samples are quickly nullified as the number of traces increases.

3.6 Countermeasures

Side-channel power analysis, as mentioned in the introduction, cannot be entirely prevented due to the power consuming nature that characterize any operating VLSI device: "*[...] to introduce direction into a transition between states, energy must be lost irreversibly. A system that conserves energy cannot make a transition to a definite state and thus cannot make a decision (compute)*" as stated by [29] and recalled by [30].

Although not completely removable, leakage from a microprocessor can be reduced and side-channel analysis can be made so complex as to make it economically unattractive, leading the attacker to spend more resources than he/she can gain from a successful attack.

In the case of SPA, for instance, implementations shall make use of constant time execution paths, avoiding the use of conditional branches when treating secret data. If possible, hardware and software engineers shall cherry-pick primitive instructions known to leak less information to implement security-critical code sections [5].

[4] To achieve similar results with DPA 2500+ traces are needed.

5.5.1) No Window - Global Case

```
cnt_found = 0
for i in range(num_keys):
    if (recovered_key[i] == known_key[i]):
        print(f"Byte #{i}:\tKey: 0x{recovered_key[i]:02X} | Expected: 0x{known_key[i]:02X} | ✅")
        cnt_found += 1
    else:
        print(f"Byte #{i}:\tKey: 0x{recovered_key[i]:02X} | Expected: 0x{known_key[i]:02X} | ❌")

print("\n")
print(f"🔵 Found {cnt_found}/16 keys! 🔵")
```

```
Byte #0:        Key: 0x2B | Expected: 0x2B | ✅
Byte #1:        Key: 0x7E | Expected: 0x7E | ✅
Byte #2:        Key: 0x15 | Expected: 0x15 | ✅
Byte #3:        Key: 0x16 | Expected: 0x16 | ✅
Byte #4:        Key: 0x28 | Expected: 0x28 | ✅
Byte #5:        Key: 0xAE | Expected: 0xAE | ✅
Byte #6:        Key: 0xD2 | Expected: 0xD2 | ✅
Byte #7:        Key: 0xA6 | Expected: 0xA6 | ✅
Byte #8:        Key: 0xAA | Expected: 0xAB | ❌
Byte #9:        Key: 0xF7 | Expected: 0xF7 | ✅
Byte #10:       Key: 0x15 | Expected: 0x15 | ✅
Byte #11:       Key: 0x88 | Expected: 0x88 | ✅
Byte #12:       Key: 0x09 | Expected: 0x09 | ✅
Byte #13:       Key: 0xCF | Expected: 0xCF | ✅
Byte #14:       Key: 0x4F | Expected: 0x4F | ✅
Byte #15:       Key: 0x3C | Expected: 0x3C | ✅


🔵 Found 15/16 keys! 🔵
```

Figure 3.12: The results obtained with a CPA attack on `AES-128`, done with just 50 traces collected with *ChipWhisperer*™ against the target `XMEGA` microcontroller: 15/16 bytes were correctly recovered.

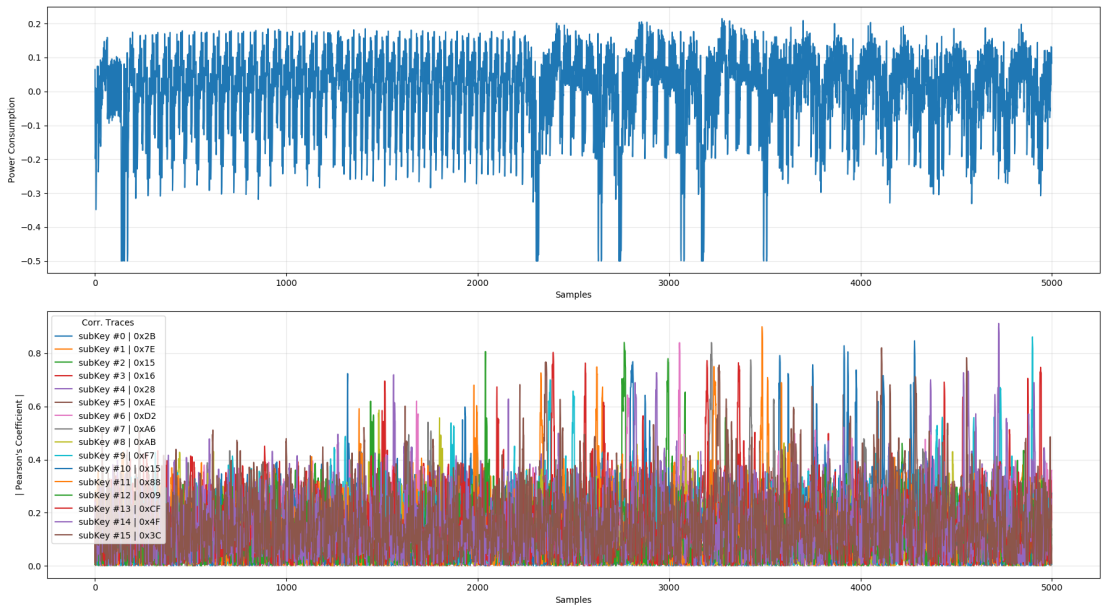

Figure 3.13: The correlation traces related to the same attack depicted in Fig. 3.12

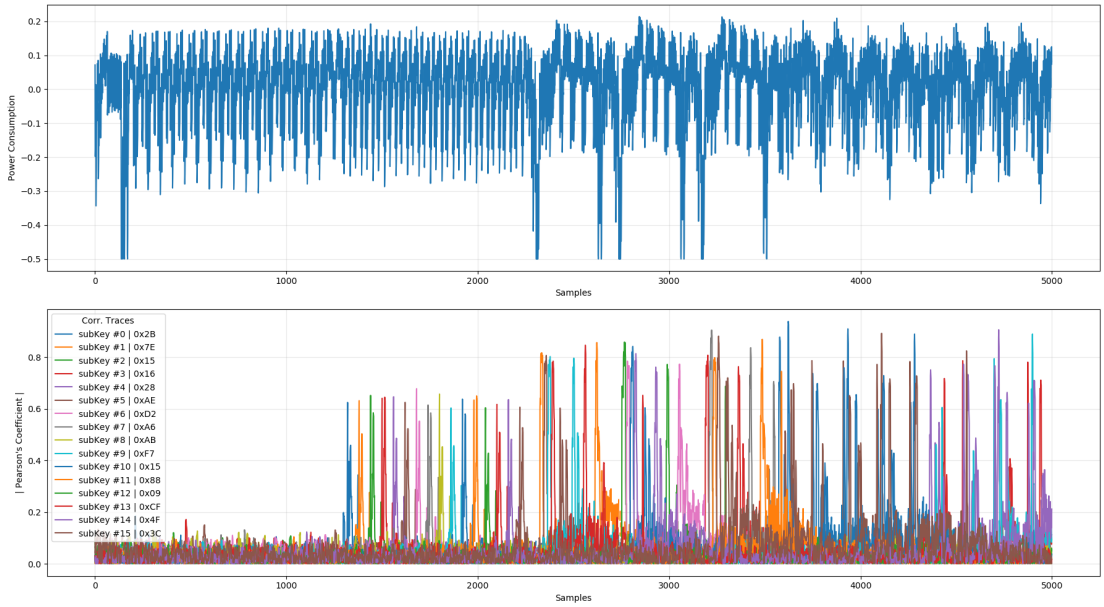

Figure 3.14: The correlation traces related to another CPA attack on `AES`, this time completed with 800 traces.

Preventing DPA and CPA (and in general all the variants based on statistical observations) requires more effort than SPA, since the effects of noise and uncorrelated device activity decrease by simply increasing the number of traces captured. In general, the countermeasures against this class of analysis adhere to two main principles:

1. reduce as much as possible the information leaked;
2. mask and "scramble" the remaining portion of the leakage, so as to make unfeasible a statistical analysis.

3.6.1 Reducing the leakage

The first point can be achieved by decreasing the *signal-to-noise* ratio: this will result in the need for a higher number of traces to obtain a successful attack. In the extreme case, this number can be made so high so that the attacker cannot afford to spend the time required for the related computations.

Balancing Balancing the power consumption reduces the observed amplitude of data dependent variations. Several solutions have been proposed such as the use of dual-rail pre-charge logic or the use of multi-bit data representations and balanced transitions [5].

Noise introduction Noise can be introduced on both the temporal and amplitude domain. The former can be introduced by using variable clocks and the insertion of random wait state and/or random branching. The latter can be achieved by leveraging the presence of circuitry able to perform computations in parallel to the main cryptographic core, adding their spikes contribution to the shared power rail. In general, if "noise" is positively correlated to the intermediate data, the attack gets easier. Conversely, if noise is negatively correlated to intermediates, the result is a balancing countermeasure. A reduction factor of k in the *signal-to-noise* ratio achieved by the addition of amplitude

noise increases the number of traces required by a factor of k^2. Temporal noise is instead less effective as the number of traces required increases linearly in k: this is because each individual trace provides $O(k)$ more locations to perform the attack.

In any case, the effectiveness of both noise components can be mitigated by advanced trace alignment and filtering techniques [5].

3.6.2 Blinding and masking

The goal of blinding and masking is to ensure that information leakage is not directly related to the secret data. This kind of countermeasure is able to resist DPA and CPA by periodically changing the random representation of secret parameters. This allows to obtain a limited amount of collected traces with the same secret representation, de facto limiting the applicability of statistical computations. The effects caused on the attack are important: the attacker has now to target the relationship between masks and masked data, relying on the so called *high-order* DPA, which requires a way higher number of traces with respect to classical DPA [5].

3.6.3 Protocol level countermeasures

Different countermeasures can also be adopted outside the implementation level, like on the cryptographic one or even closer to the application level. As stated by Kocher *et al.* "*The most effective and least difficult way to address side channel attacks is to design cryptographic protocols to survive leakage*" [5]. The general approach is to limit the number of cryptographic operations performed with a single key: even in the case of a successful side-channel attack, the effect of subsequent attacks can be limited by simply periodically changing the key. In a similar way, a cryptanalyst can assess how many bits per transaction the implementation is leaking. Knowing the desired security level for the application (expressed in bits), it can define the key cryptoperiod in transaction terms, enforcing an automatic key change whenever the number of transactions reaches a certain critical threshold.

3.7 Tools: The *ChipWhisperer*™ Kit

The growing popularity of side-channel analysis in recent years has driven the research for powerful, compact and low-priced tools that could compete against lab-grade instruments. The latest advances in specialized hardware are enabling the ability to perform both passive and active attacks even in the educational environment and to novices in the field, effectively further democratizing this class of attack.

One of the most interesting (and probably the most known) tool in the market is NewAE's *ChipWhisperer*™. *ChipWhisperer*™ is an entire ecosystem composed by many tools (both hardware and software) proposed for embedded hardware security research targeting, in the specific, side-channel analysis and fault injection. A single *ChipWhisperer*™ board can be considered as a viable substitute for digital oscilloscopes: the extreme flexibility and small physical size allow the attacker/cryptanalyst to easily reproduce attack setups without relying on external tools.

3.7.1 *ChipWhisperer*™ Hardware

ChipWhisperer™ delivers a capture technique that commercial oscilloscopes usually do not provide: synchronous sampling. In opposition to these instruments, where both the clock of the capturing device and the clock of the device under attack run freely, *ChipWhisperer*™ is able to derive the former from the latter in order to always measure a consistent point in time. In other cases, *ChipWhisperer*™ is also able to generate a single clock signal which is used by both the device under attack and *ChipWhisperer*™'s own sampling circuitry. Synchronous sampling allows to keep always the same phase relation among the two clocks: this relaxes the sample rate requirements with respect to asynchronous sampling. For instance, in [31], the authors present an attack against a SASEBO-GII board claiming that synchronously sampling at 96 MS/s produces similar results obtained by an asynchronous sampling done at 2 GS/s. The lower frequencies needed are mainly due to the absence of jitter and phase variations among the two clocks, something that is simply not possible to achieve if the two clocks run freely and are therefore completely independent.

Additionally, *ChipWhisperer*™ provides the necessary hardware to perform fault analysis and glitch-attacks, de facto merging a good majority of the instruments used by an hardware attacker (Fault attacks and glitching are topics not covered for the purposes of this thesis).

3.7.2 *ChipWhisperer*™ Lite Characteristics

ChipWhisperer™ capture board can leverage both asynchronous and synchronous sampling up to a frequency of 105MS/s. Being the frequency quite low (with respect to a lab-grade oscilloscope), the synchronous sampling technique is preferred. The capture circuitry leverages an AC-coupled probe, a low-noise amplifier with adjustable gain and a 10-bit ADC. The sample buffer can contain up to 24573 samples. However, x4 and x10 decimation factors are possible, allowing to cover even larger temporal ranges. *ChipWhisperer*™ leverages an FPGA to manage the entire capture and glitching phase, while counting on an Atmel microcontroller for configuration and interface with the attacker's personal computer. The board also provides built-in serial ports and programmers to communicate with daughter boards, if present.

Finally, the entire design, both hardware and software, is completely open-source, allowing the attacker to modify all the aspects of the product and to possibly contribute to its development.

ChipWhisperer™ `XMEGA` Target Board

NewAE also produces kits providing an additional evaluation board to be attacked by the main *ChipWhisperer*™ capture device. The kit used for this thesis work provides, as target board, an `ATXMEGA128D4` 8-bit microcontroller by Atmel. In Fig. 3.15 it is possible to appreciate the main *ChipWhisperer*™ attacker device (on the left) and the target detachable `XMEGA` board on the right.

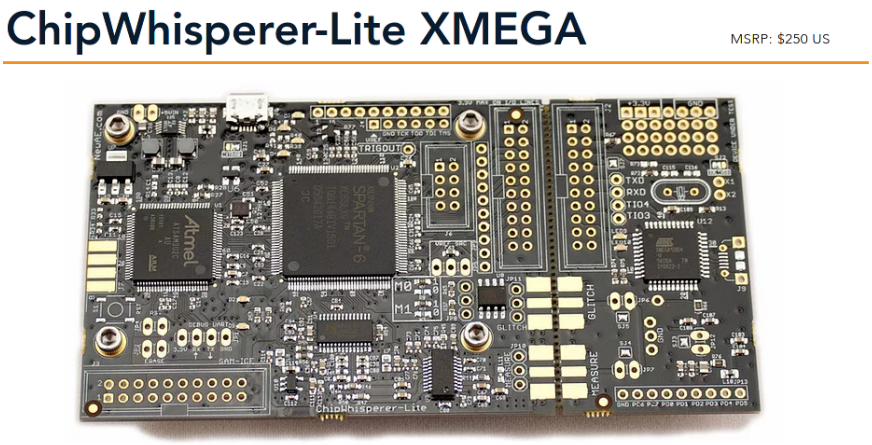

Figure 3.15: NewAE's *ChipWhisperer*™ Lite board, on the left, and the target detachable `XMEGA` board, on the right.

The two boards share the same clock signal, therefore allowing for synchronous sampling. The `XMEGA` microcontroller receives an "untouched" version of the clock, running at 7MHz, while the capture board multiplies by four the main frequency, bringing the sampling clock to 28MHz. This allows to achieve what visible in image 3.16: each clock cycle happening in the device is sampled in four different instants. Both the capture probe and glitching interconnection can be achieved, once the two boards detached, using two shielded SMA cables.

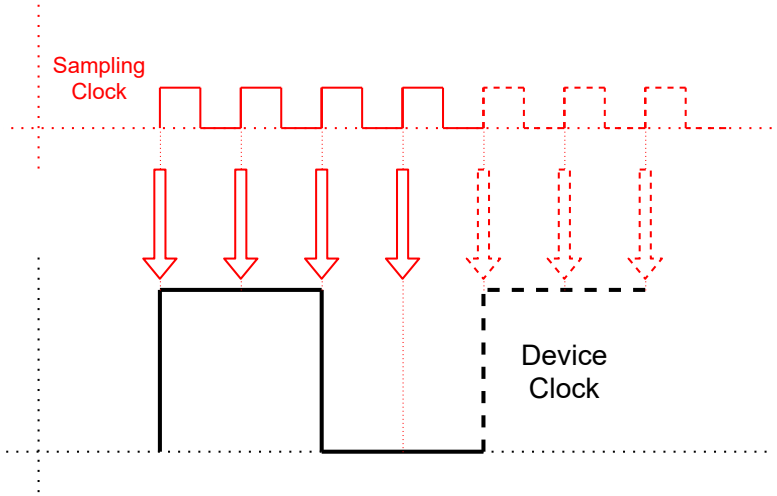

Figure 3.16: *ChipWhisperer*™'s default synchronous sampling clock vs. `XMEGA`'s (attacked device) default operating clock.

3.7.3 *ChipWhisperer*™ Sofware

The *ChipWhisperer*™ main capture board can completely substitute an oscilloscope: the samples collected during the acquisition are automatically sent to the attacker's personal computer. The software interface between the board and the PC is achieved thanks to a set of Python API's. The exposed commands allow to deeply configure all the capture parameters of the board, such as the decimation factors, the adjustable gain, the sampling clock and many others. Once the board correclty configured, the attacker can launch an acquisition phase using a Python script and continue with the offline statistical analysis within the same script. The entire client-side API architecture, being completely based on Python, enables the attacker the use for the extremely large library of additional modules provided by the language, such as `numpy`, `pandas` and `matplotlib`. The extreme portability of the device and the flexibility provided by Python allows for fast, scriptable and highly reproducible setups, allowing to ease the work of the generic cryptanalyst and enabling a low-cost approach to hardware security in academic and educational environments.

Chapter 4

S-Box Variants: State of the Art

The development of countermeasures against side-channel attacks is a continuously evolving field of the academic research. Their importance is becoming crucial, especially given the extreme efficiency these attacks can reach with respect to classical cryptanalysis. For instance, while differential power analysis can leverage as just as 2000 bytes of plaintext-ciphertext pairs, linear or differential cryptanalysis needs terabytes of such pairs to achieve the same result [14]. In general, for a malicious actor, the choice of which attack to use is often dictated by how practical the attack is: in the case of an embedded device this unquestionably falls on side-channel attacks.

The problem of finding proper countermeasures presents a multitude of approaches and trade-offs to be evaluated. Since each different implementation addresses different security needs, it is therefore important for both the researcher and the hardware/software engineer to have adequate knowledge of the possible solutions that can be implemented.

In general, the following scenario can be observed: the greater the security provided by a particular solution, the greater its impact on the device performance. As an example, we can cite the two following solutions:

- **Temporal Noise Addition**: provides a lightweight solution able to increase the overall complexity of a side-channel attack with relatively small overhead on the algorithm complexity. However, the random delays inserted can be easily recognized and circumvented thanks to the use of inference techniques such as the one proposed in [32];
- **Masking Schemes**: provides a strong security against SCA (side-channel analysis) but induces a high performance overhead, making its implementation not practicable in small embedded devices [33] [34].

While some implementations may accept a significant performance impact to achieve a higher level of security, others, however, must necessarily settle for a lower level of security in order to keep performance (or power consumption) within certain parameters. For instance, in the case of smartcards, even lighter solutions can significantly impact the entire project design. In general, the use of countermeasures at the implementation level has been claimed to impact both performance and code size roughly by a factor of two, directly damaging the entire area of embedded cryptography where the computation power and the memory capacity are limited [35].

In recent years the focus has therefore shifted to solutions that could abandon the implementation level and move towards the same logical and mathematical level of algorithmic specifications, as briefly proposed by Kocher in [5]. The main motivation behind this choice considers resistance to side-channel attacks in the same way as resistance

to classical cryptanalysis, treating it as a key parameter to be taken into account for the successful development of old and new block cipher algorithms. As stated by Prouff in [35] "*the design of DPA-resistant algorithms would make the addition of countermeasures unnecessary. Such a design could be done by selecting pertinent S-boxes.*"

4.1 Introduction on Novel S-Box Implementations

Many of the recent proposals are directly targeting S-Box designs. Since these structures already provide many properties against classical cryptanalysis, it is therefore reasonable to integrate in their design possible countermeasures against side-channel attacks [36].

Recent research has already proven the importance of S-Boxes against SCA. Boey *et al.* observed a significant correlation among their use in multiple block ciphers and the feasibility of a side-channel attack against their hardware implementations. For instance, the authors noticed a higher resistance against CPA in DES algorithm than in AES, proven by the higher number of traces needed to recover the round key. The observations have been related to the fact that DES leverages the presence of multiple unique S-Boxes with a lower signal-to-noise ratio in comparison to other algorithms. A further reason is due to the fact that DES S-Boxes implement a non-bijective 6-to-4 bits mapping, de facto shrinking the Hamming weight distribution at their output.

4.1.1 Theoretical metrics against classical cryptanalysis

To define the properties and qualities of an S-Box design many parameters have been defined over time. With respect to classical cryptanalysis, five basic properties are used in literature [37]:

- **Bijection**: An S-Box $S(n, m)$ is said to be bijective if it maps each input $x \in B^n$ to a distinct output $y = S(x) \in B^m$ and all possible 2^m outputs are present. An S-Box $S(n, m)$ can be bijective only when $n = m$. A bijective S-Box $S(n, n)$ represents a permutation of its 2^n inputs, since each input is mapped to a distinct output and all possible 2^n outputs are present. In this way, $S(n, n)$ will be reversible, that is, there is a mapping from each distinct output to its corresponding input;

- **Nonlinearity**: To test this property the S-Box is firstly expressed in terms of linear equations and then evaluated using *Walsh* spectrum. The nonlinearity value of AES S-Box is 112;

- **Strict Avalanche Criterion - SAC**: A Boolean function is said to satisfy the *Strict Avalanche Criterion* if each of the output bits change with a probability $p = \frac{1}{2}$ whenever a single bit is complemented at the input [38]. As a consequence, a slight change in the input leads to a large change in the output (an avalanche effect), effectively making it difficult for an attacker to infer the input value from the output. The optimum value for the *SAC* criterion is 0.5 [37];

- **Bit Independence Criterion - BIC**: This property states that two output bits j and k should change independently when any single input bit i is complemented, for all i, j and k;

- **Input/Output XOR Distribution**: As mentioned in 2.3.3, differential cryptanalysis is based on the use of the imbalances in the Differential Distribution Table (DDT). If an S-Box has an equiprobable input/output XOR distribution, the S-Box is immune to a differential attack. In the case of AES, the maximum value of differential propagation probability is of 4/256 (or 2^{-6}).

4.1.2 Theoretical metrics against side-channel analysis

Fairly recent publications also proved an opposition among properties associated to classical cryptanalysis and the feasibility of side-channel attacks. For instance, Guilley *et al.* proved that the more an `S-Box` is protected against linear cryptanalysis, the more vulnerable it is to side-channel attacks such as DPA [39]. Similarly, Prouff in [35] points out that a trade-off between the classical cryptographic criteria and resistance to DPA attacks has to be found, since their respective properties are in opposition.

As for classical cryptanalysis, new properties have been defined over the years to address and formally define the resistance of `S-Box`es against side-channel analysis:

- **Transparency Order**: Formally defined by Prouff in [35], the Transparency Order (TO) can quantify the resistance of an `S-Box` to DPA attacks: the smaller the TO of an `S-Box` the higher its resistance. The author also proved that the lower bound for the TO (meaning the highest possible resistance against DPA) is achieved by particular affine functions, whereas the upper bound is achieved by bent functions[1] (proving his initial claim that the higher the nonlinearity of a Boolean function the lower its resistance against DPA). Carlet in [14] defines a lower bound for this metric, proving that highly nonlinear Boolean functions like the ones that compose the `S-Box` of `AES` have "very bad transparency orders";

- **Improved Transparency Order**: Following the work of Prouff on the transparency order, Chakraborty *et al.* define, a decade later, the so called Improved Transparency Order. The authors prove that Prouff's definition is not sufficient to achieve a satisfying level of security. However, choosing `S-Box`es with small minimum transparency order is a sound strategy if combined with other classical countermeasures like masking. Based on the definition given by the authors, the lower the ITO, the lower the success probability to extract the secret key based on leakages associated to the `S-Box` [41];

- **Confusion Coefficient**: In 2012, Fei *et al.* [42] introduced the Confusion Coefficient (CC). According to the authors, a low (standard) deviation of the confusion coefficient κ characterizes an `S-Box` with a high resistance against physical attacks. However, Picek *et al.* [43] as well as *Stoffelen* [44] report that the higher the CC metric, the higher the resistance of the `S-Box` against side-channel attacks [45]. The confusion coefficient measures the discrepancy between the hypothesis of an intermediate state using the correct (secret) key and any hypothesis made with a (wrong) key assumption [36]. Therefore, as one compares possible intermediate processed values, the confusion coefficient depends on the underlying cryptographic algorithm and thus, if the attacker targets an `S-Box` operation, on the side-channel resistance of that specific `S-Box` [36];

- **Autocorrelation Function**: The maximum value AC_{max} assumed by this parameter, obtained when evaluating all the Boolean functions that compose an `S-Box` structure, can be considered as a measure of the resistance of the `S-Box` against SCA. The lower AC_{max} the better [46];

- **Success Rate**: Also known as *success probability*, the SR is used by cryptanalysts to measure the resistance of an implementation against CPA. The SR metric represents the probability with which a physical attack can extract secret keys from the device [45].

[1] Bent functions are a special type of Boolean functions that present resistance *by definition* to linear cryptanalysis and apparently perfect resistance against differential cryptanalysis. [40]

A work by Carlet, Heuser and Picek [36] highlights the fact that the (almost) preservation of Hamming weight and a small Hamming distance between a plaintext x and an intermediate $F(x)$ are two properties which, from an intuitive perspective, can strengthen the resistance to SCA. The authors also verify the claim and additionally observe that, especially in the case of the Hamming weight, S-Boxes with no difference between the HW of their input and output have nonlinearity equal to 0, therefore obtaining terrible properties against classical cryptanalysis. The authors finally confirm that having an S-Box more resilient against SCA will make it potentially more vulnerable against classical cryptanalysis, without however excluding possible trade-offs, similarly to what Prouff already proved.

A first publication to prove the theoretical claims advanced on many new S-Box structures was addressed by Lerman *et al.* in [45]. The authors analyzed several designs, highlighting the fact that, in some cases, metrics proposed above like CC and TO lack accuracy to evaluate the resistance of S-Boxes against side-channel attacks.

For instance, when theoretically evaluating the designs under attack, the authors observed that both TO and CC metrics contradict themselves. For instance, in the case of larger S-Boxes, where a higher nonlinearity is expected, the authors obtained a large value of the TO which also indicates a lower resistance to SCA, therefore confirming Prouff's claim [35] on the opposition of the two properties. However, Lerman's observations did not replicate the same rationale in the case of the Confusion Coefficient, leading to a first contradiction between the two metrics. The results obtained with theoretical simulation also indicated that theoretical metrics such as TO and CC *"do not match actual attacks when the leakage model matches the leakage function (representing the worst case scenario i.e., when the adversary knows how the device leaks information)."* Finally, when evaluating experimental results on a real device, the authors concluded that *"the complexity of a side-channel attack cannot be represented using any theoretical metrics based on a single scalar value"* and that *"the outcomes of the theoretical metrics do not always reflect the success rate of a side-channel attack when considering simulated and real leakages"* [45].

4.2 Design Methodologies

The most common generation methods used for the design of S-Box structures can be divided into three main categories: algebraic construction, pseudo-random generation and heuristic techniques.

Algebraic methods have been used to build many "historical" S-Box structures, such as the one used in AES [16]. They are in fact known for their excellent cryptographic properties against linear and differential cryptanalysis, thus making them the preferred choice by cryptographers in past years. Further analysis has shown that this methodology leads to the existence of a system of polynomial equations with low degree which can pose as a "potential vulnerability" of the cipher to algebraic attacks [47].

In the case of pseudo-random generation, S-Boxes are constructed from a table of random values. This methodology warrants random structures, but the S-Boxes generated lack of relevant cryptographic properties in most cases [48].

Finally, the heuristic approach is an attempt to maintain the randomness present in the structure of pseudo-random generated S-Boxes, with properties as close as possible to the ones present in algebraic constructions [48]. This methodology will be further elaborated later.

In cryptography, it is also not uncommon to build an S-Box from smaller ones, usually an 8-bit S-Box from several 4-bit S-Boxes. In many cases, such a structure is used not

only to allow an efficient implementation of the S-Box in hardware or using a bit-sliced approach, but also to protect S-Boxes implemented in this way against side-channel attacks [47].

In recent years, novel design methodologies like chaos-based generation, quantum-based structures and many others have been presented. The resulting S-Box structures claim to overcome both classical and side-channel cryptanalysis. In the following sections a brief overview on the most common methodologies is presented, illustrating the corresponding results and publications.

4.2.1 Chaos-Based Methods

Chaos-based methodologies propose a new rationale to be used for the creation of S-Box structures, based on the use of chaotic systems as a source of randomness. The three main reasons behind this choice are summarized by Özkaynak in [49]:

- Chaotic systems' long-term behavior is non-periodic;
- System trajectories are dependent on the initial conditions and control parameters;
- The equation of a chaotic system is deterministic and its irregular behavior is caused by intrinsic nonlinearities rather than noise.

The above characteristic present a good premise for structures able to provide enough *confusion* to a block cipher algorithm, like S-Boxes.

Many different randomness sources have been presented in literature, some of which leverage chaotic maps and hyper chaotic systems (see [50], [51], [52] and [53]). For instance, Özkaynak in [49] produced more than 20.000 S-Box based on discrete and continuous-time chaotic systems as Logistic Map and Lorenz's system. In most cases, the construction of S-Boxes with good cryptographic properties involves the use of optimization algorithms to operate on the initial values provided by the main source of randomness. The choice of a good source of randomness is extremely important, so are its initial parameters. The values it returns need to be normalized in the range $[0, 255]$ and possibly discarded in case the final value obtained is already in the S-Box LUT (to guarantee bijection). Once a large set of possible S-Box candidates is available, the optimization function iterates on every candidate building a ranking and sorting the best candidates which maximizes both nonlinearity and differential propagation (also called I/O XOR distribution).

The same publication by Özkaynak demonstrated that current chaos-based S-Box structures have worse cryptographic properties than those of the S-Box proposed by Daemen and Rijmen for AES. For instance, while the nonlinearity index of AES S-Box is 112 [16], in the case of chaos-based designs the maximum value achieved has been shown to be 106.75 [49]. Similarly, while the maximum differential propagation probability in the case of AES S-Box is of 4/256 (even if most entries achieve even lower ratios: either 0/256 or 2/256 [16]), the **minimum** value for this metric achieved by current chaos-based structures is of 10/256 [49]. The above metrics, also highlighted in a derivative work by Açikkapi in [37], are sufficient to prove that the actual chaos-based S-Box structures are less resistant to classical cryptanalysis such as linear and differential attacks.

In literature, as reported in [37], chaos-based S-Boxes have been indicated as an alternative defense against implementation attacks, especially side-channel analysis. In this publication, the authors tested 3 different S-Box structures: the first being the classical one used in AES while the remaining two are chaos-based and directly taken from Özkaynak's research [49] (the authors picked the best and the worst performing S-Boxes proposed in his paper). The results obtained prove that the proposed chaos-based

S-Box structures are *slightly* more resistant against side-channel attacks. However, by recalling the statement given by Prouff in [35] we can argue back with the observations obtained by Açikkapi, claiming that the observed strengths against SCA are mainly due to the low nonlinearity and low differential probability score obtained by both chaos-based structures.

4.2.2 Heuristic Methods

Heuristic methods fall into the class of optimization algorithms: their goal is to solve a search problem in a faster and more efficient way with respect to exact/exhaustive algorithms, sacrificing, in general, the quality of the final results for speed of execution. In most cases, due to the fact that many problems have an NP-Hard complexity, heuristic algorithms are de facto the only feasible approach to solve them. For instance, in the case of an 8x8 S-Box structure like the one of AES, the different possible S-Boxes obtainable are 256^{256} (since each of the 256 input values can assume 256 different output values): the search space is incredibly large [54].

Many heuristic algorithms have been proposed for the search of S-Box structures with good trade-offs between classical cryptanalysis properties and SCA resistance. In 2014 Picek *et al.* [43] leveraged heuristic methods and the confusion coefficient CC to build S-Boxes with improved resistance to implementation attacks. In the case of 8x8 S-Boxes the improvement have, once again, the counter-effect of worsening classical cryptanalytical properties like nonlinearity and differential uniformity.

In general, as observed by Picek *et al.* in [55], the quality of solutions produced by such methods could fit somewhere between the random search (which return S-Boxes with poor cryptographic results) and algebraic constructions (which return structures having the best cryptographic properties). Moreover, the authors highlight the importance of exploring the entire space of solutions. Since many cryptographic properties are in opposition between each other, moving away from algebraic methodologies and explore the use of heuristic algorithms is beneficial as it allows to explore the available trade-offs. This claim confirms the possibility for heuristic methods to improve properties related to side-channel attacks providing good trade-offs with cryptographic properties. In this same publication, the authors present a new cost function, used to obtain bijective S-Boxes that reach a nonlinearity scores of 104 [55].

In [46] the authors leverage both genetic algorithm (GA) and a hill-climb method to design S-Boxes with relatively high nonlinearity and low autocorrelation. The result obtained also prove that the heuristic algorithm used can find S-Box structures faster than exhaustive search, with better cryptographic properties with respect to the ones obtained with random search.

One of the latest contributions to heuristic methodologies has been proposed by Freyre *et al.* in 2020 [48]. The authors adopted a novel hybrid heuristic method based on *Leaders and Followers*[2] and *hill-climbing* algorithms obtaining "excellent results for confusion coefficient variance". In addition, the group leverages the cost function proposed by Picek in [55] to improve the nonlinearity score of their findings, obtaining values always greater or equal than 100. The authors run their algorithm on a Haswell *Core™-i7* , obtaining two of the most performing S-Box among the entire set they built in 40 and 44 hours of running time. The structure obtained prove to have better **theoretical** side-channel resistance than many real-life S-Boxes used in algorithms like AES and PICARO [34] while satisfying nonlinearity and differential properties.

[2] A metaheuristic algorithm that tries to avoid biased comparisons among the evolved results by using two distinct populations: the first stores the best solutions found (leaders), the second (followers) stores sub-optimal solutions pursuing the results obtained by the first group. [48]

Chapter 5

Experimental Results

5.1 Introduction and motivations

The work carried out in this thesis consists in empirically verifying the resilience to side-channel attacks of some of the latest proposals of `S-Box` structures. The results obtained provide an idea of the qualities of such structures, allowing to understand the relevance of the results achieved by the academic research. The objective is to test the structures examined in the context of a real attack scenario carried out by means of specialized but low cost and affordable equipment.

We have chosen to analyze a series of chaos-based and heuristic-based `S-Box` structures produced by the respective methodologies, considered among the most promising.

Similar discussions have already been addressed in several recently published papers. In 2011 Boey, O'Neil and Woods [56] tested four different ASIC implementations (including `AES` and `DES`) on a SASEBO board. In 2016 Lerman *et al.* [45] proposed an extensive attack testing multiple `8x8` structures on an *ATmega 328p* 8-bit microcontroller, carried on using an external oscilloscope. The authors observed that the obtained experimental results differed from the outcome of theoretical metrics. Finally, in 2019 Açikkapi *et al.* tested two different chaos-based `S-Box`es initially proposed by Özkaynak in [49]. The setup included a *ChipWhisperer*™ Lite board attacking an `XMEGA` 8-bit microcontroller. The authors compared the results obtained against the ones related to the original `AES S-Box` proving the presence of a slightly higher resistance to side-channel analysis.

5.2 `S-Box`es overview: properties and selection criteria

The attack is conducted onto six different `S-Box` structures, one of which is the original `AES S-Box`, used as a reference structure. The software implementation of the `AES-128` algorithm in use is comparable to what is currently used in smartcards and other small embedded devices to protect user data in everyday scenarios.

The remaining five `S-Box` structures under attack were selected among the latest publications for both chaos-based and heuristic-based methodologies. Three of these come from the work presented by Freyre *et al.* in [48] and were designed to specifically target resistance to side-channel analysis.

The fifth `S-Box` selected corresponds to the last structure among the six proposed by Hussain *et al.* in [57], referred in the original publication to as "S-box-6". The authors leverage the use of a chaotic logistic map as a source of randomness in order to achieve a structure with excellent properties against cryptanalysis. The methodology used, however, does not take into account any design criteria to improve resistance to side-channel analysis. This `S-Box` is thus considered in the same way as the original structure of `AES`

S-Box	Nonlinearity	Differential Uniformity	Confusion Coefficient
sbox_aes [16] [43]	112.0	4/256	0.111
sbox_freyre_1 [48]	100.0	8/256	4.500
sbox_freyre_2 [48]	100.0	8/256	4.492
sbox_freyre_3 [48]	102.0	8/256	1.934
sbox_hussain_6 [57]	112.0	4/256	n.d.
sbox_ozkaynak_1 [49]	106.7	10/256	n.d.

and it is compared to all other designs provided with theoretical countermeasures against SCA.

The last structure taken into consideration has been initially proposed by Özkaynak in [49] and tested by Açikkapi in [37] with a similar attack setup as the one proposed in this thesis. The structure, in the original publication, is referred to as "Proposal 2".

The cryptographic properties of the six structures under attack are presented in Table 5.2, whereas their hexadecimal representation is available at `https://github.com/Mrcuve0/Thesis-Work`.

5.3 Instrumentation setup

This thesis work leverages a similar instrument setup to the one used by Açikkapi in [37]. The attack is in fact carried on using a *ChipWhisperer*™ Lite, whose kit also provides an `XMEGA` target board (see 3.7). The `XMEGA` 8-bit `AVR` microcontroller runs the `AES-128` software implementation under attack. The entire communication with the attacked microcontroller happens thanks to the *ChipWhisperer*™ board, which also acts as a programmer. The data transmitted to the microcontroller, such as the plaintext and the secret key, is firstly crafted on a personal computer. It is then sent to the *ChipWhisperer*™ capture board which finally forwards it to the `XMEGA` board via serial communication. The `AES-128` implementation is therefore wrapped by a user program able to listen on a serial port for data transmitted. The entire `XMEGA` code is ready to use and provided by NewAE itself [58].

5.4 Steps of the analysis

The proposed analysis consists of evaluating each of the selected structures individually, changing the code implementation of `AES-128` at each iteration, replacing the standard `S-Box` structure with the one under attack. The substitution is automatically achieved at compilation time. The microcontroller is then programmed with the modified version of the code leveraging the main *ChipWhisperer*™ board, which acts as an AVR programmer.

Once the target board programmed, the *ChipWhisperer*™ is instructed to capture a certain number of traces. For each trace the Python script running on the personal computer creates a new plaintext, which is then sent together with the (fixed) encryption key to the capture board. The board then resets the microcontroller, sends the received `[plaintext - key]` pair and launches an `AES-128` execution on the microcontroller side, while capturing the related power trace. Finally, the power trace data is sent back to the personal computer. The process is repeated N times, where N indicates the number of traces to be collected. Once the capture phase is complete, the microcontroller code is compiled again by swapping the current structure with the next structure to be evaluated. The entire process is repeated until there are no more `S-Box`es to evaluate.

Once the trace collection completed, *ChipWhisperer*™ can be disconnected and the attack can proceed on the personal computer for offline analysis, which consists of a correlation power attack. The analysis is iterated, once again, for a number of times equal to the number of different structures to analyze. For each structure, a reduced replica of the first steps of `AES-128` (`addRoundKey` and `SubBytes`) is built. The leakage model adopted defines the estimate of the power consumption as the Hamming weight of the intermediate value obtained as output of the replica. All the N traces collected are analyzed following the standard methodology used for correlation power attacks (see 3.5).

The implementation of the attack is already provided as a built-in Python function provided by *ChipWhisperer*™ (also known as "chipwhisperer analyzer"). Some modifications are although necessary to adapt the leakage model in use: as the `S-Box` in the microcontroller changes at each iteration during the capture phase, the same must happen with the leakage model during the analysis phase.

The provided APIs provide the possibility to periodically pause the attack after the evaluation of a certain number of traces. Having this possibility allows the attacker to build a custom Python callback function which enables for a multitude of purposes. In the most common scenario, this function can be used to report the current status of the attack, observing the intermediate results achieved. In our specific implementation, the callback function allows to save multiple "checkpoints" of the results obtained: each time the attack is paused the results are dumped in a Pandas `DataFrame` structure. This data structure allows to easily handle the various results in a multi-row multi-column table format, enabling for a fast and simplified data management. Each checkpoint defines a new `DataFrame` instance, which is slightly manipulated to highlight the (possible) presence of the correct key bytes in a specific point of the attack. The final table structure obtained ranks the possible values of each of the 16 subkeys from most to least likely. The number r of rows in the table (representing the related r subkey guesses) is user-defined. Conversely, the number of columns is fixed to 16, the size of an entire key. Each structure is finally automatically exported as a pre-compiled LaTeX table and to `html` and `csv` files.

Once the offline analysis for the current `S-Box` is completed, all the data and results obtained during the various checkpoints are used to draw two different plots:

1. **PGE v. Traces plot**: this graph shows the *PGE* (Partial Guessing Entropy) trend of each of the correct subkeys. The guessing entropy is generally defined as *"the average number of successive guesses required with an optimum strategy to determine the true value of a random variable X"* [59] [60]. In this case, as highlighted by O'Flynn and Chen in [60], the guessing entropy represents the distance, in terms of positions, of the correct subkey value from the top position in the ranking made from all the possible subkey candidates. In other terms the PGE graph shows, for each checkpoint, how far the correct value of the subkey is from the first position in the ranking. Since the collected results data grows as the number of completed checkpoints increases, we expect the PGE of each subkey to reduce overtime, meaning that the more traces are analyzed the more the correct subkey candidate is assumed to be found. In the case of `S-Box` structure with side-channel resistance properties we expect an additional difficulty in finding the correct subkey candidates, meaning that the PGE obtained by each subkey should be greater than 0 even for a high number of traces.

2. **Correlation v. Traces plot**: this graph shows the correlation trend for all the 16 subkey bytes superimposed to the correlation of the wrong key candidates. The value represented in the graph corresponds to the absolute value of Pearson's linear correlation coefficient, meaning that the range value spans from 0, the minimum, to

1, the maximum. An absolute correlation of 1 indicates that the leakage model in use perfectly represents the real power consumption of the device. In a standard correlation power attack the correlation graph gradually splits and isolates the correct subkey candidates from the wrong ones, meaning the attack is able to successfully reconstruct the secret key used. On the other hand, in the case of `S-Box`es with side-channel resistance properties, we expect the attack is unable to satisfyingly separate the correct key candidates form the wrong ones. The graph should result in a tangled representation of the various correlations.

5.5 Results and comments

The analysis illustrated in the previous section has been implemented using a Python script, able to automatically handle all the aspects related to the data collection: from the `S-Box` code manipulation to the automatic graph creation. The script source code, the data and the graphs obtained are available at `https://github.com/Mrcuve0/Thesis-Work`.

In this section are reported the results obtained when running the main script for 50, 100, 1000 and 5000 traces. Each time the six different `S-Box` structures were tested, collecting the results and plotting the related graphs.

As expected, the original `AES S-Box` proved to be weak against correlation power analysis, revealing nearly the whole key after just 30 different executions. As pictured in Fig 5.1, 15 out of the 16 subkeys were entirely revealed at the 40-th trace. At this checkpoint, 15 subkeys had a $PGE = 0$, meaning that the corresponding key candidates were (correctly) leading the ranking. Byte #8, however, entered the $PGE < 10$ zone only after 55 traces. This means that, prior 55 traces, 10 incorrect subkey candidates for Byte #8 had a higher correlation, therefore were in higher position in the ranking for that specific byte position. The $PGE < 10$ threshold has been proposed by O'Flynn and Chen in [60] as a valuable indicator of the resistance of an implementation against side-channel analysis. This threshold should be taken into consideration only when the last byte of the key crosses it, entering the zone below the threshold. If such scenario happens, all the 16 bytes can be found within 10 places in the related ranking, meaning that an attacker can easily retrieve the key with a reduced exhaustive search even if not all key bytes found have a $PGE = 0$. Although being quite simplistic, this indicator can be adopted to denote at which point a specific `S-Box` can be considered broken.

Table 5.1 reports the various instants in which each `S-Box` broke the $PGE < 10$ threshold. The results are reported for all the four executions of the Python script. The data collected moving from one execution to the other (e.g., from 50 traces to 100 traces) is not additive. This means that, for instance, the second execution captured 100 traces *ex novo*, without accounting for the 50 traces captured in the first execution. This explains the differences among different executions for the same `S-Box`. It is easily observable that the margin of error provided by this indicator is quite large, since some `S-Box`es break the threshold with a difference of 20 traces moving from one execution to another. However, in two cases, `sbox_freyre_1` and `sbox_freyre_2`, the resistance against the PGE threshold is quite remarkable, largely above a possible margin of error.

Similar results to the `AES S-Box` were reported by the structures proposed by Hussain and Özkaynak, with a slightly improved but no significant resistance for the latter. The results obtained by analyzing Hussain's `S-Box` confirmed that no side-channel countermeasure was accounted during the structure design phase, as expected. For the sake of brevity. the graphs related to these two structures are not included as their representations (and overall results) are similar to the ones related to the `AES S-Box`. Likewise, the first and the second `S-Box` proposals by Freyre obtains similar results in all the scenarios tested.

S-Box	50 traces	100 traces	1000 traces	5000 traces
sbox_aes	≈ 32 traces	≈ 55 traces	below th.	below th.
sbox_freyre_1	over th.	over th.	≈ 190 traces	below th.
sbox_freyre_2	over th.	over th.	≈ 520 traces	below th.
sbox_freyre_3	≈ 43 traces	≈ 26 traces	below th.	below th.
sbox_hussain_6	≈ 39 traces	≈ 46 traces	below th.	below th.
sbox_ozkaynak_1	over th.	≈ 49 traces	below th.	below th.

Table 5.1: How many traces where required to break the $PGE < 10$ threshold for each attack iteration?

Therefore, we decided to report, among the two, only the second structure.

Conversely, we decided to report the sbox_freyre_3 structure presented by Freyre, since it shows some similar properties to sbox_freyre_2 but with a much lower amplitude. For instance, the structure reached the PGE threshold in less traces than AES S-Box, as visible in 5.5, denoting no significant improvements against side-channel analysis. In 5.6 the "confusion" added to the correlation traces is, however, clearly visible, especially if directly compared with the same graph in the case of AES 5.2. In any case, we can see that the "entanglement" produced in the correlation traces is not sufficient to prevent the attack, as there is still a clear separation between the two groups of traces.

The results obtained by the second S-Box structure proposed by Freyre *et al.*, sbox_freyre_2, can be appreciated in Fig 5.3 and Fig 5.4, respectively. If we compare these two plots with the ones obtained by the original AES S-Box, in Fig 5.1 and Fig 5.2, we can easily observe how the structure proposed by Freyre significantly improves the resistance against SCA. For instance, in the PGE graph 5.3 we can observe how Byte #12 and Byte #8 fluctuates multiple times without entering with an appreciable momentum the subthreshold zone. The correlation plot 5.3 shows how tangled the correlation traces of the correct subkeys are with the ones related to wrong guesses, with the correlations of Byte #12 and Byte #8 separating from the rest of the group of correct key candidates.

Moving towards the third and fourth executions of the Python script (1000 and 5000 traces), we can further appreciate the separation among the various correlations in graphs like Fig 5.7 and Fig 5.8. Both graphs in 5.8 and in 5.10 converge into three main sets of traces, with the one with higher correlations associated to the set of correct key guesses. In the case of sbox_freyre_2in Fig 5.9, both Byte #12 and Byte #8 continue to present lower correlations with respect to the other correct subkeys. This fact, however, does not significantly impact the results for the PGE indicator, which threshold has already been broken many traces before, at around 500 traces.

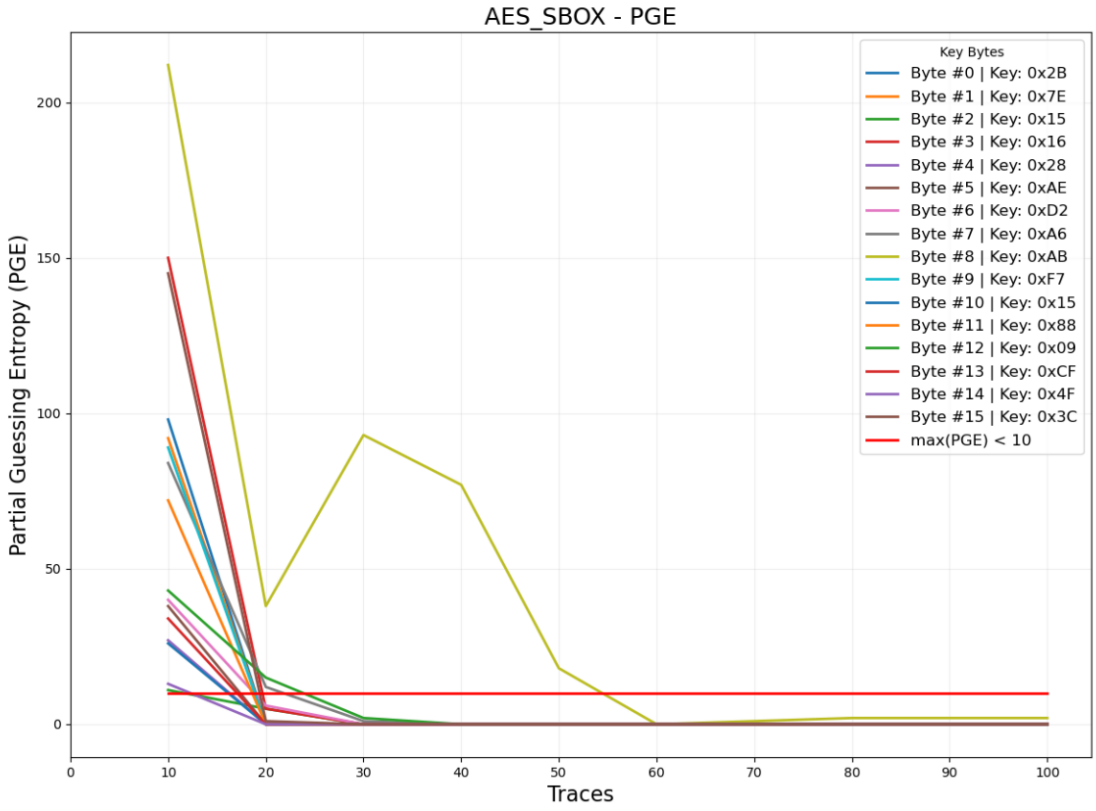

Figure 5.1: PGE for `AES S-Box`, 100 traces, PGE threshold crossed at ≈ 55 traces.

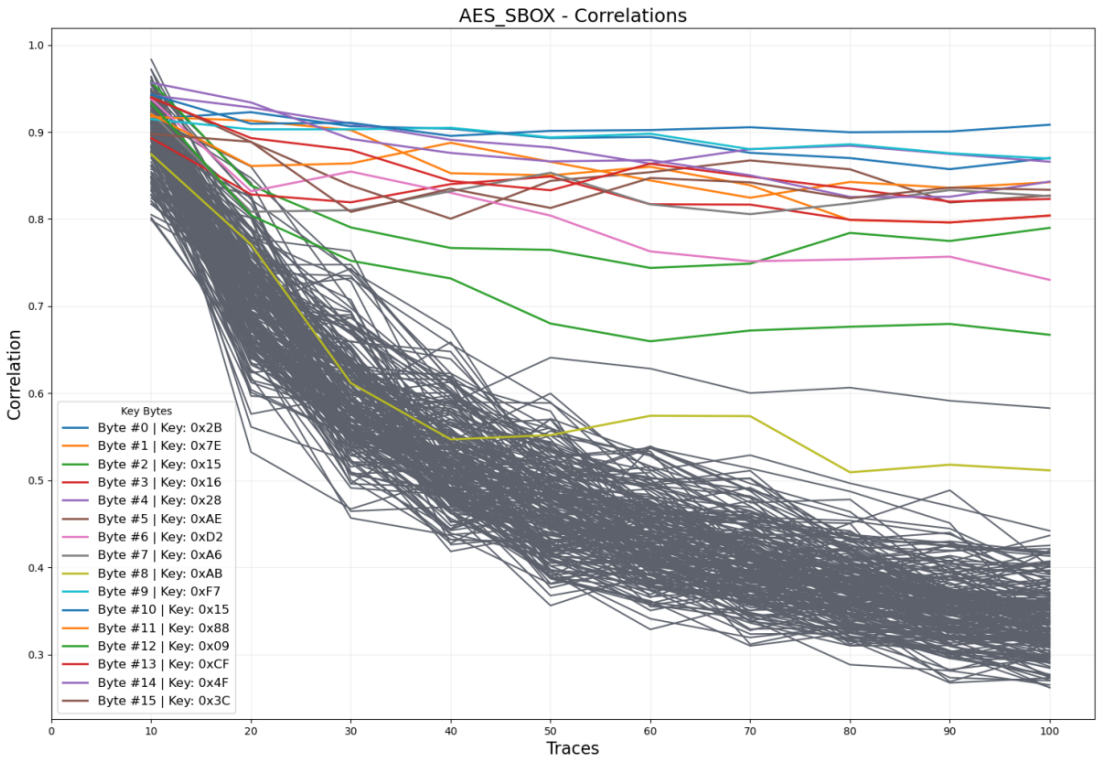

Figure 5.2: Correlations for `AES S-Box`, 100 traces.

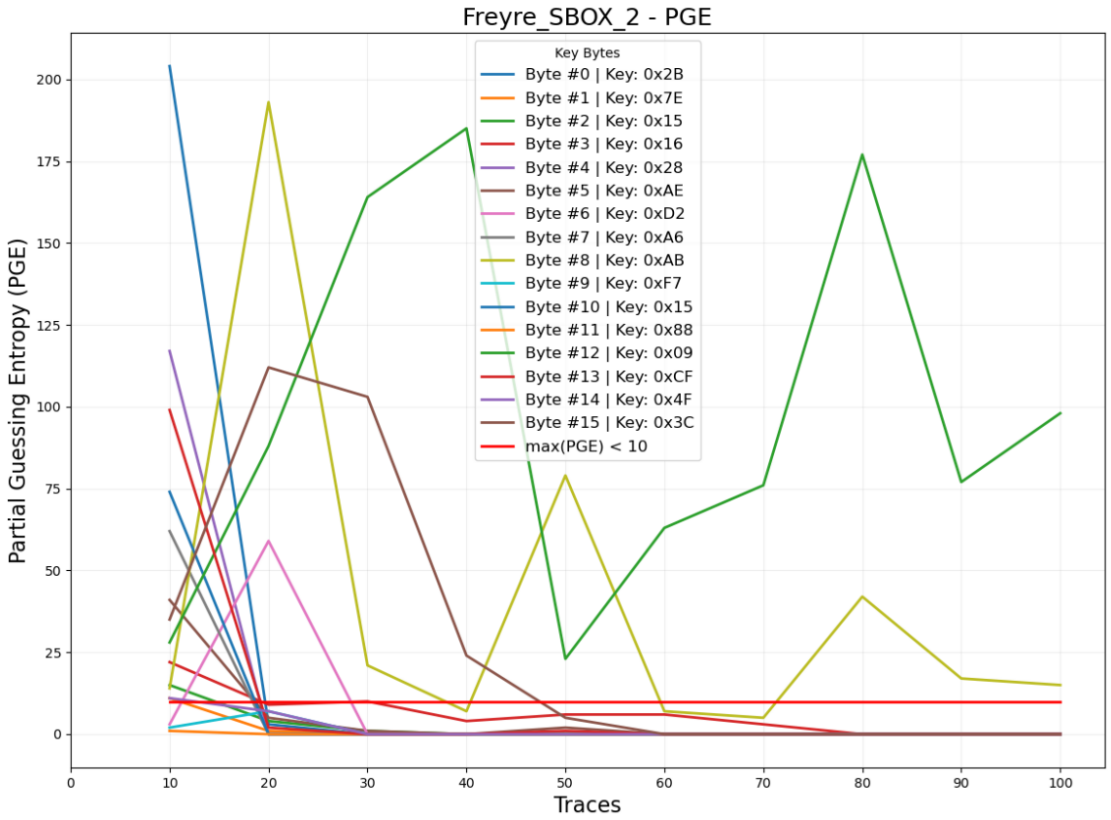

Figure 5.3: PGE for `sbox_freyre_2`, 100 traces, PGE threshold NOT crossed.

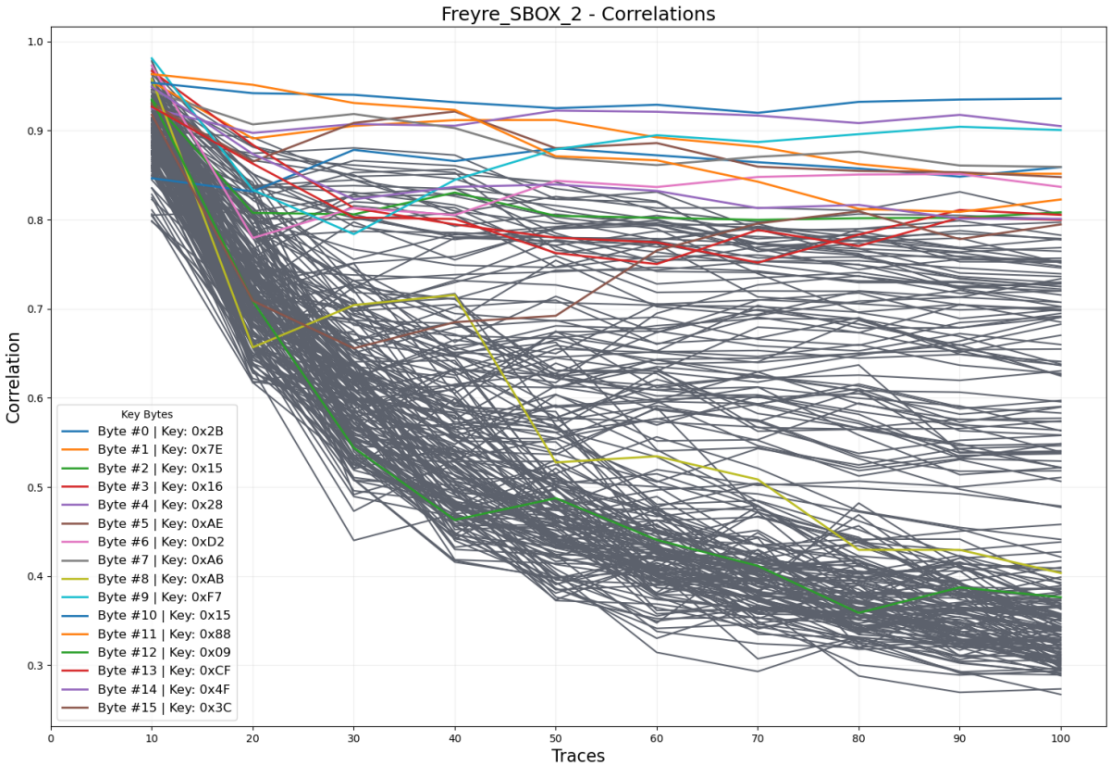

Figure 5.4: Correlations for `sbox_freyre_2`, 100 traces.

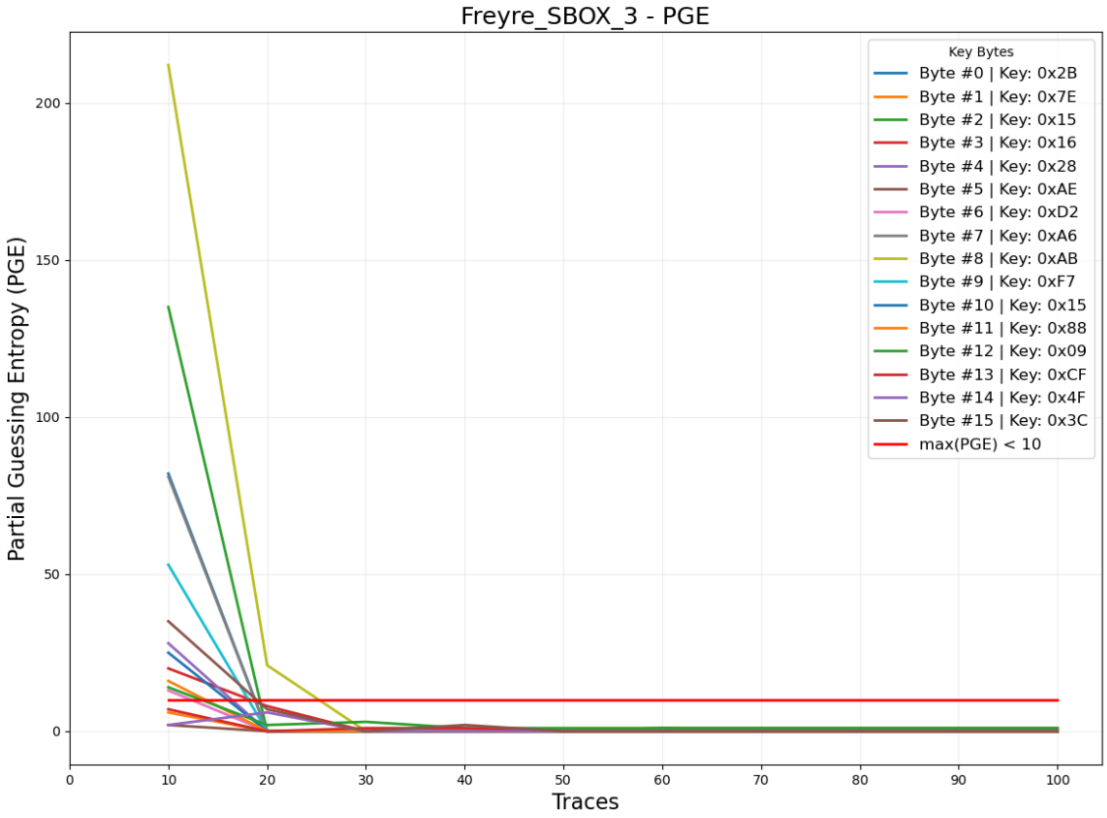

Figure 5.5: PGE for `sbox_freyre_3`, 100 traces, PGE threshold crossed at ≈ 27 traces.

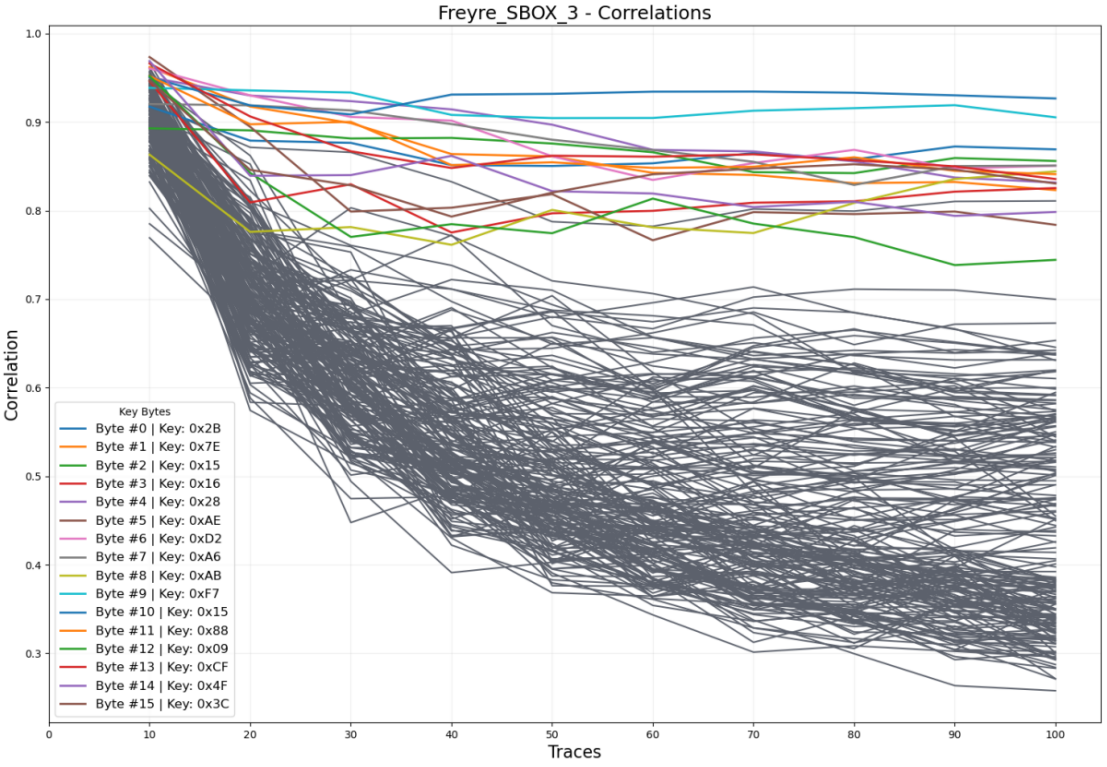

Figure 5.6: Correlations for `sbox_freyre_3`, 100 traces.

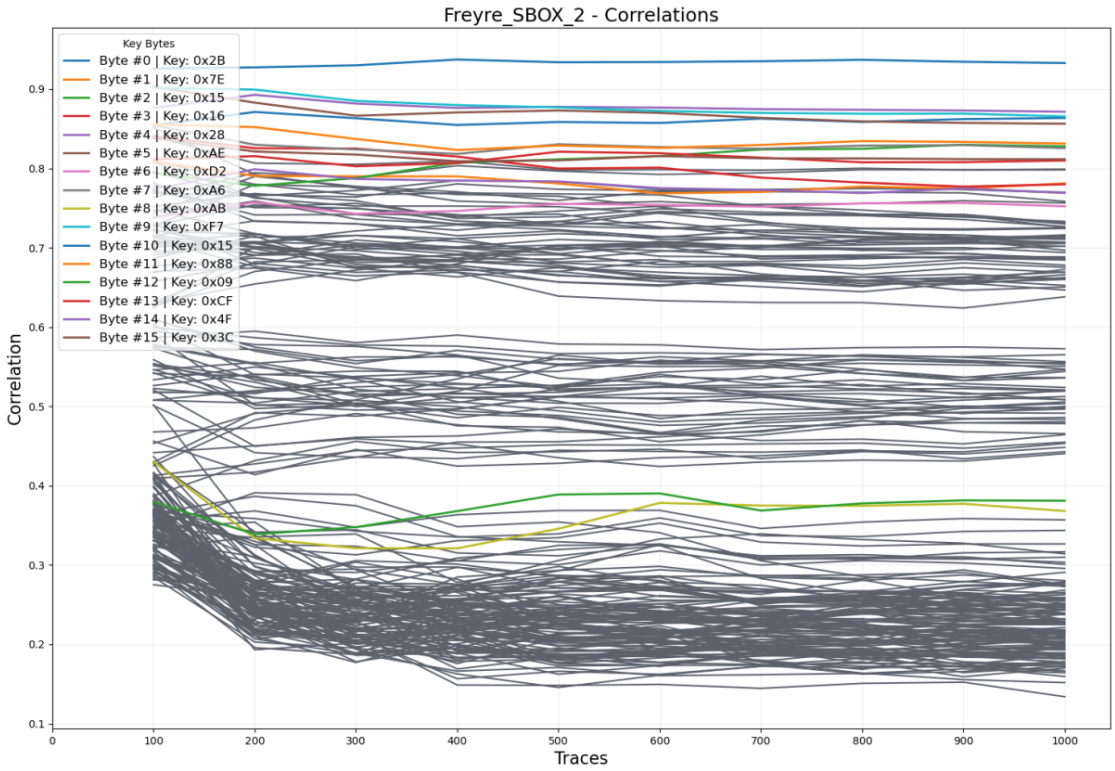

Figure 5.7: Correlations for `sbox_freyre_2`, 1000 traces.

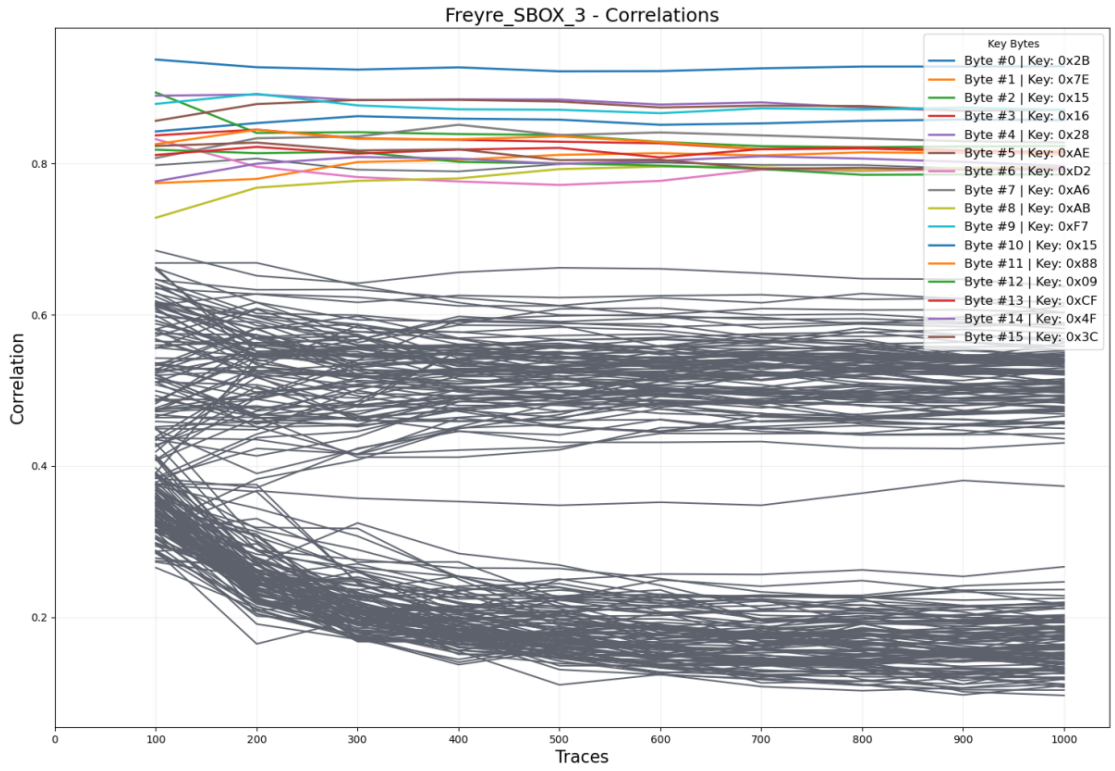

Figure 5.8: Correlations for `sbox_freyre_3`, 1000 traces.

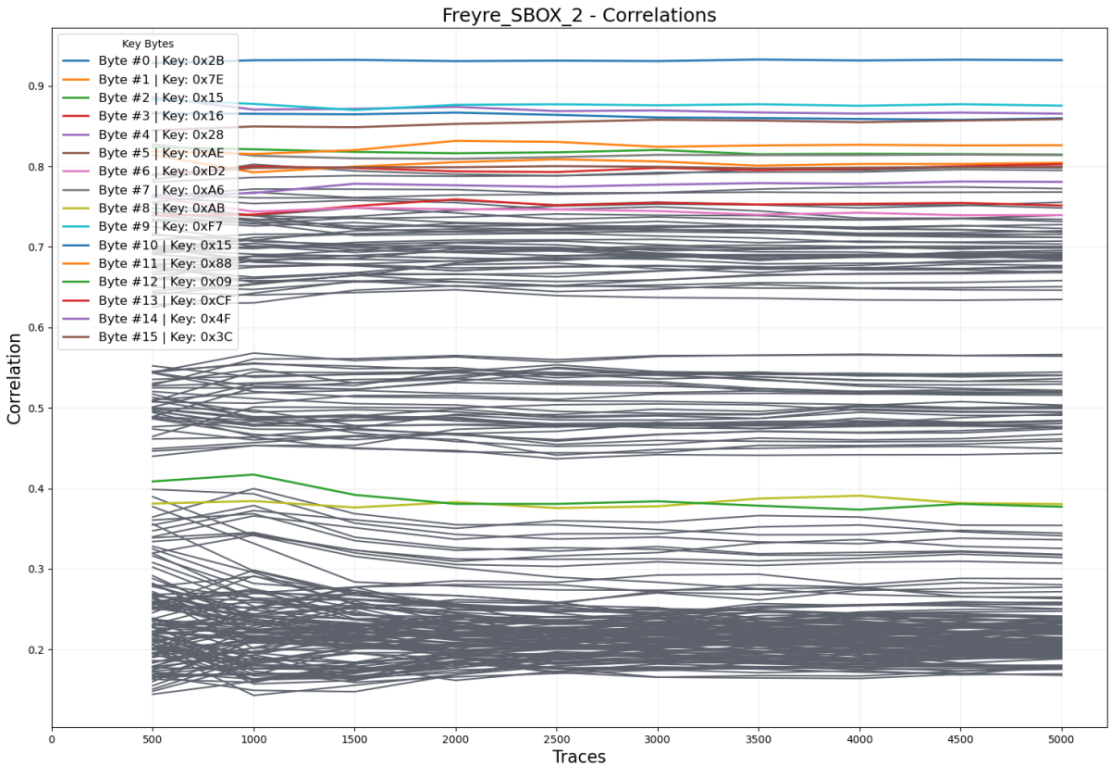

Figure 5.9: Correlations for `sbox_freyre_2`, 5000 traces.

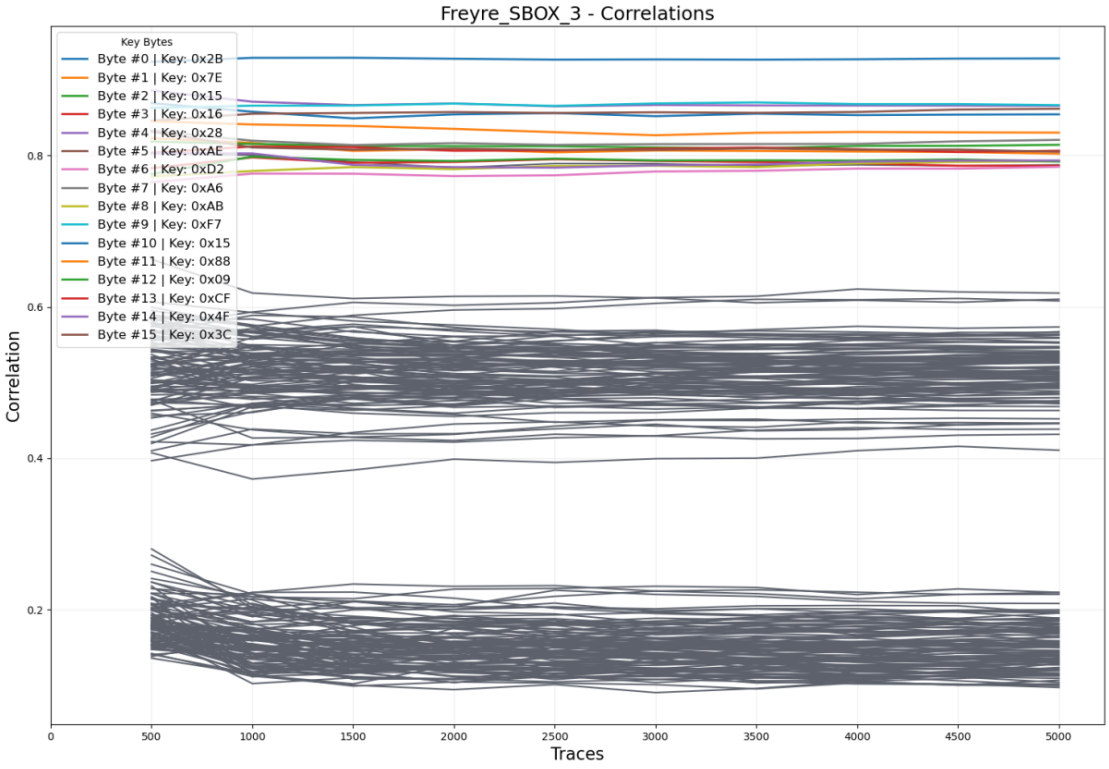

Figure 5.10: Correlations for `sbox_freyre_3`, 5000 traces.

Chapter 6

Conclusions

6.1 Comments

The results obtained and partially discussed in the previous chapter confirm the ability of some of the latest `S-Box` proposal to provide a significantly higher resistance to side-channel analysis, acting as a lightweight countermeasure against this class of attacks.

The attack procedure carried on in this thesis confirmed once again the poor performance provided by the original `AES S-Box` structure. These kind of structures, created without taking into account side-channel analysis, fail to provide any additional protection. The `S-Box` proposed by Hussain in [57] was selected on purpose to confirm the theory already demonstrated by Prouff, Guilley, Picek and many others: properties like nonlinearity and differential uniformity do not relate to the presence of any natural countermeasure against SCA.

Other chaotic structures like the one proposed by Özkaynak in [49] provide slightly better results if compared against the `S-Box` of `AES` but still not sufficient to successfully prevent a physical attack, pushing the number of required traces not too far away from what originally done by the original encryption algorithm.

Finally, the heuristic-based structures proposed by Freyre *et al.* in [48] provide great insight on the state of the art achieved by this design methodology. The graphs obtained in the previous chapter demonstrates the highest side-channel resistance among all the `S-Box`es tested, in some cases increasing the trace requirements up to a factor of 10 5.1. The metrics provided by Freyre enable for further observations on the impact of the Confusion Coefficient. From the data collected it is possible to notice that the higher the Confusion Coefficient the higher the number of traces required and the lower the discrepancy among the various correlation traces, like pictured in graphs like 5.4 5.7 and 5.9. The observed data perfectly confirm the claims provided by Picek *et al.* in [43] and *Stoffelen* in [44]. We observed that both `sbox_freyre_1` and `sbox_freyre_2` structures acted similarly, having a CC of 4.5 and 4.492, respectively. Conversely, `sbox_freyre_3`, with a CC of 1.934, showed some principle of correlation confusion but not significant enough to increase considerably the resistance against side-channel power analysis.

6.2 Future work

Future work may be directed toward the analysis of additional new `S-Box` structures, with the ultimate goal of gaining an even broader view of the state of the art in side-channel resistance designs. Further real-world testing is still needed as the demand for lightweight countermeasures continues to increase.

Further testing will need to be conducted leveraging the same physical configuration, with the goal of defining empirical metrics that can better indicate resistance against physical attacks than the PGE metric used in our tests.

Finally, a comprehensive characterization capable of correlating theoretical metrics to real-world results is needed, expanding the analysis to additional properties such as transparency order and revisited transparency order.

Bibliography

[1] Matt Crypto. SubBytes operation for AES. Available at `https://it.wikipedia.org/wiki/File:AES-SubBytes.svg`, August 2006. Accessed 13 March 2021.

[2] Matt Crypto. ShiftRows operation for AES. Available at `https://it.wikipedia.org/wiki/File:AES-ShiftRows.svg`, August 2006. Accessed 12 March 2021.

[3] Matt Crypto. MixColumns operation for AES. Available at `https://it.wikipedia.org/wiki/File:AES-MixColumns.svg`, August 2006. Accessed 12 March 2021.

[4] Matt Crypto. AddRoundKey operation for AES. Available at `https://it.wikipedia.org/wiki/File:AES-AddRoundKey.svg`, August 2006. Accessed 12 March 2021.

[5] Paul Kocher, Joshua Jaffe, Benjamin Jun, and Pankaj Rohatgi. Introduction to differential power analysis. *Journal of Cryptographic Engineering*, 1(1):5–27, 2011.

[6] Michal Varchola and Milos Drutarovsky. The differential power analysis laboratory setup. In *Proceedings of 22nd International Conference Radioelektronika 2012*, pages 1–4. IEEE, 2012.

[7] Kazuo Sakiyama, Yu Sasaki, and Yang Li. *Security of block ciphers: from algorithm design to hardware implementation*. John Wiley & Sons, 2016.

[8] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, et al. Spectre attacks: Exploiting speculative execution. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1–19. IEEE, 2019.

[9] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, et al. Meltdown: Reading kernel memory from user space. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 973–990, 2018.

[10] Paul Röttger and Artur Janc. A Spectre proof-of-concept for a Spectre-proof web. Available at `https://security.googleblog.com/2021/03/a-spectre-proof-of-concept-for-spectre.html`, March 2021. Accessed 15 March 2021.

[11] R. Baldoni, R. De Nicola, and P. Prinetto. *Il Futuro della Cybersecurity in Italia: Ambiti Progettuali Strategici*. Consorzio Interuniversitario Nazionale per l'Informatica - CINI, 2018. ISBN: 9788894137330.

[12] Paolo Prinetto and Gianluca Roascio. Hardware security, vulnerabilities, and attacks: A comprehensive taxonomy. In *ITASEC*, pages 177–189, 2020.

[13] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Annual international cryptology conference*, pages 388–397. Springer, 1999.

[14] Claude Carlet. On highly nonlinear s-boxes and their inability to thwart dpa attacks. In *International Conference on Cryptology in India*, pages 49–62. Springer, 2005.

[15] Hans Dobbertin, Lars Knudsen, and Matt Robshaw. The cryptanalysis of the aes–a brief survey. In *International Conference on Advanced Encryption Standard*, pages 1–10. Springer, 2004.

[16] Joan Daemen and Vincent Rijmen. Aes proposal: Rijndael. 1999.

[17] Christof Paar and Jan Pelzl. *Understanding cryptography: a textbook for students and practitioners*. Springer Science & Business Media, 2009.

[18] Christophe De Canniere, Alex Biryukov, and Bart Preneel. An introduction to block cipher cryptanalysis. *Proceedings of the IEEE*, 94(2):346–356, 2006.

[19] Martin Hellman. A cryptanalytic time-memory trade-off. *IEEE transactions on Information Theory*, 26(4):401–406, 1980.

[20] Eli Biham and Adi Shamir. Differential cryptanalysis of the full 16-round des. In *Annual International Cryptology Conference*, pages 487–496. Springer, 1992.

[21] Howard M Heys. A tutorial on linear and differential cryptanalysis. *Cryptologia*, 26(3):189–221, 2002.

[22] Tomoyasu Suzaki, Kazuhiko Minematsu, Sumio Morioka, and Eita Kobayashi. Twine: A lightweight block cipher for multiple platforms. In *International Conference on Selected Areas in Cryptography*, pages 339–354. Springer, 2012.

[23] Mitsuru Matsui. Linear cryptanalysis method for des cipher. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 386–397. Springer, 1993.

[24] Mitsuru Matsui. The first experimental cryptanalysis of the data encryption standard. In *Annual International Cryptology Conference*, pages 1–11. Springer, 1994.

[25] Atmel. Atmel 8 and 16 bit avr microcontrollers - atxmega. Available at `http://ww1.microchip.com/downloads/en/devicedoc/atmel-8135-8-and-16-bit-avr-microcontroller-atxmega16d4-32d4-64d4-128d4_datasheet.pdf`, September 2016. Accessed 15 March 2021.

[26] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In *International workshop on cryptographic hardware and embedded systems*, pages 16–29. Springer, 2004.

[27] Wikipedia Community. Pearson Correlation Coefficient - Wikipedia. Available at `https://en.wikipedia.org/wiki/Pearson_correlation_coefficient`. Accessed 26 March 2021.

[28] Colin O'Flynn. Correlation Power Analysis - NewAE Wiki. Available at `https://wiki.newae.com/Correlation_Power_Analysis`. Accessed 26 March 2021.

[29] Carver Mead and Lynn Conway. Introduction to vlsi systems, 1980.

[30] Jean-Sébastien Coron, David Naccache, and Paul Kocher. Statistics and secret leakage. *ACM Transactions on Embedded Computing Systems (TECS)*, 3(3):492–508, 2004.

[31] Colin O'Flynn and Zhizhang Chen. A case study of side-channel analysis using decoupling capacitor power measurement with the openadc. In *International Symposium on Foundations and Practice of Security*, pages 341–356. Springer, 2012.

[32] Jean-Sébastien Coron and Ilya Kizhvatov. Analysis and improvement of the random delay countermeasure of ches 2009. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 95–109. Springer, 2010.

[33] Suresh Chari, Charanjit S Jutla, Josyula R Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In *Annual International Cryptology Conference*, pages 398–412. Springer, 1999.

[34] Gilles Piret, Thomas Roche, and Claude Carlet. Picaro–a block cipher allowing efficient higher-order side-channel resistance. In *International Conference on Applied Cryptography and Network Security*, pages 311–328. Springer, 2012.

[35] Emmanuel Prouff. Dpa attacks and s-boxes. In *International Workshop on Fast Software Encryption*, pages 424–441. Springer, 2005.

[36] Claude Carlet, Annelie Heuser, and Stjepan Picek. Trade-offs for s-boxes: Cryptographic properties and side-channel resilience. In *International Conference on Applied Cryptography and Network Security*, pages 393–414. Springer, 2017.

[37] Mehmet Şahin Açikkapi, Fatih Özkaynak, and Ahmet Bedri Özer. Side-channel analysis of chaos-based substitution box structures. *IEEE Access*, 7:79030–79043, 2019.

[38] Réjane Forrié. The strict avalanche criterion: spectral properties of boolean functions and an extended definition. In *Conference on the Theory and Application of Cryptography*, pages 450–468. Springer, 1988.

[39] Sylvain Guilley, Philippe Hoogvorst, and Renaud Pacalet. Differential power analysis model and some results. In *Smart Card Research and Advanced Applications Vi*, pages 127–142. Springer, 2004.

[40] Wikipedia Community. Bent function - Wikipedia. Available at `https://en.wikipedia.org/wiki/Bent_function`. Accessed 29 March 2021.

[41] Kaushik Chakraborty, Sumanta Sarkar, Subhamoy Maitra, Bodhisatwa Mazumdar, Debdeep Mukhopadhyay, and Emmanuel Prouff. Redefining the transparency order. *Designs, codes and cryptography*, 82(1-2):95–115, 2017.

[42] Yunsi Fei, A Adam Ding, Jian Lao, and Liwei Zhang. A statistics-based success rate model for dpa and cpa. *Journal of Cryptographic Engineering*, 5(4):227–243, 2015.

[43] Stjepan Picek, Kostas Papagiannopoulos, Barış Ege, Lejla Batina, and Domagoj Jakobovic. Confused by confusion: Systematic evaluation of dpa resistance of various s-boxes. In *International Conference on Cryptology in India*, pages 374–390. Springer, 2014.

[44] Ko Stoffelen. *Intrinsic side-channel analysis resistance and efficient masking*. PhD thesis, Master's thesis, Radboud University, 2015.

[45] Liran Lerman, Olivier Markowitch, and Nikita Veshchikov. Comparing sboxes of ciphers from the perspective of side-channel attacks. In *2016 IEEE Asian Hardware-Oriented Security and Trust (AsianHOST)*, pages 1–6. IEEE, 2016.

[46] William Millan, L Burnett, Gary Carter, Andrew Clark, and Ed Dawson. Evolutionary heuristics for finding cryptographically strong s-boxes. In *International Conference on Information and Communications Security*, pages 263–274. Springer, 1999.

[47] Reynier Antonio de la Cruz Jiménez. Generation of 8-bit s-boxes having almost optimal cryptographic properties using smaller 4-bit s-boxes and finite field multiplication. In *International Conference on Cryptology and Information Security in Latin America*, pages 191–206. Springer, 2017.

[48] Alejandro Freyre-Echevarría, Ismel Martínez-Díaz, Carlos Miguel Legón Pérez, Guillermo Sosa-Gómez, and Omar Rojas. Evolving nonlinear s-boxes with improved theoretical resilience to power attacks. *IEEE Access*, 8:202728–202737, 2020.

[49] Fatih Özkaynak. Construction of robust substitution boxes based on chaotic systems. *Neural Computing and Applications*, 31(8):3317–3326, 2019.

[50] Miroslav M Dimitrov. On the design of chaos-based s-boxes. *IEEE Access*, 8:117173–117181, 2020.

[51] Khawaja Muhammad Ali and Majid Khan. Application based construction and optimization of substitution boxes over 2d mixed chaotic maps. *International Journal of Theoretical Physics*, 58(9):3091–3117, 2019.

[52] Guoping Tang, Xiaofeng Liao, and Yong Chen. A novel method for designing s-boxes based on chaotic maps. *Chaos, Solitons & Fractals*, 23(2):413–419, 2005.

[53] Fatih Özkaynak. Chaos based substitution boxes as a cryptographic primitives: Challenges and opportunities. *Chaotic Model. Simul.*, 1:49–57, 2019.

[54] John A Clark, Jeremy L Jacob, and Susan Stepney. The design of s-boxes by simulated annealing. *New Generation Computing*, 23(3):219–231, 2005.

[55] Stjepan Picek, Marko Cupic, and Leon Rotim. A new cost function for evolution of s-boxes. *Evolutionary computation*, 24(4):695–718, 2016.

[56] Kean Hong Boey, Maire O'Neill, and Roger Woods. How resistant are sboxes to power analysis attacks? In *2011 4th IFIP International Conference on New Technologies, Mobility and Security*, pages 1–6. IEEE, 2011.

[57] Iqtadar Hussain, Tariq Shah, Muhammad Asif Gondal, and Hasan Mahmood. An efficient approach for the construction of lft s-boxes using chaotic logistic map. *Nonlinear Dynamics*, 71(1):133–140, 2013.

[58] NewAEtech. ChipWhisperer - NewAE Github repository. Available at `https://github.com/newaetech/chipwhisperer`. Accessed 29 March 2021.

[59] James L Massey. Guessing and entropy. In *Proceedings of 1994 IEEE International Symposium on Information Theory*, page 204. IEEE, 1994.

[60] Colin O'Flynn and Zhizhang Chen. Synchronous sampling and clock recovery of internal oscillators for side channel analysis and fault injection. *Journal of Cryptographic Engineering*, 5(1):53–69, 2015.